05-23-00

"Express Mail" mailing label number EL398315910US.
Date of Deposit May 22, 2000

Case No. 9623/179

PATENT APPLICATION TRANSMITTAL LETTER

To the Assistant Commissioner for Patents:

Transmitted herewith for filing is the patent application of: Phillip G. Rorex, Thomas A. Soulanille and Bradley R. Haugaard for

METHOD AND APPARATUS FOR IDENTIFYING RELATED SEARCHES IN A DATABASE SEARCH SYSTEM. Enclosed are:

☒ six (6) sheet(s) of drawings, 28 pages of application (including title page), and the following Appendices : thirty eight (38) page Source Code Appendix.

☐ Declaration.

☐ Power of Attorney.

☐ Verified statement to establish small entity status under 37 CFR §§ 1.9 and 1.27.

☐ Assignment transmittal letter and Assignment of the invention to : _____.

☐ _____.

Claims as Filed	Col. 1	Col. 2
For	No. Filed	No. Extra
Basic Fee		
Total Claims	25-20	5
Indep. Claims	4-3	1
Multiple Dependent Claims Present		

*If the difference in col. 1 is less than zero, enter "0" in col. 2.

Small Entity			Other Than Small Entity	
Rate	Fee	or	Rate	Fee
	$ 345	or		$ 690
x$9=	$	or	5x$18=	$90
x$39=	$	or	1x$78=	$78
+$130=	$	or	+$260=	$0
		or		
Total	$		Total	$858

☐ Please charge my Deposit Account No. 23-1925 in the amount of $: _____. A duplicate copy of this sheet is enclosed.

☐ A check in the amount of $: _____ to cover the filing fee is enclosed.

☐ The Assistant Commissioner is hereby authorized to charge payment of the following fees associated with this communication or credit any overpayment to Deposit Account No. 23-1925. A duplicate copy of this sheet is enclosed.

- ☐ Any additional filing fees required under 37 CFR § 1.16.
- ☐ Any patent application processing fees under 37 CFR §1.17.

☐ The Assistant Commissioner is hereby authorized to charge payment of the following fees during the pendency of this application or credit any overpayment to Deposit Account No. 23-1925. A duplicate copy of this sheet is enclosed.

- ☐ Any filing fees under 37 CFR § 1.16 for presentation of extra claims.
- ☐ Any patent application processing fees under 37 CFR § 1.17.
- ☐ The issue fee set in 37 CFR § 1.18 at or before mailing of the Notice of Allowance, pursuant to 37 CFR § 1.311(b).

May 22, 2000
Date

John G. Rauch
BRINKS HOFER GILSON & LIONE
Registration No. 37,218

"Express Mail" mailing label number EL398315910US

Date of Deposit: May 22, 2000

Our Case No. 9623/179

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
APPLICATION FOR UNITED STATES LETTERS PATENT

INVENTOR: Phillip G. Rorex
Thomas A. Soulanille
Bradley R. Haugaard

TITLE: METHOD AND APPARATUS FOR IDENTIFYING RELATED SEARCHES IN A DATABASE SEARCH SYSTEM

ATTORNEY: John G. Rauch
BRINKS HOFER GILSON & LIONE
P.O. BOX 10395
CHICAGO, ILLINOIS 60610
(312) 321-4200

METHOD AND APPARATUS FOR IDENTIFYING RELATED SEARCHES IN A DATABASE SEARCH SYSTEM

APPENDIX / COPYRIGHT REFERENCE

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the U.S. Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

An Appendix of computer program source code is included herewith. The Appendix is hereby expressly incorporated herein by reference, and contains material which is subject to copyright protection as set forth above.

BACKGROUND OF THE INVENTION

The present invention relates generally to a method and system for generating a search result list, for example, using an Internet-based search engine. More particularly, the present invention relates to a method and system for generating search results from a pay for performance database and generating a list of related searches from a related search database.

Search engines are commonly used to search the information available on computer networks such as the World Wide Web to enable users to locate information of interest that is stored within the network. To use a search engine, a user or searcher typically enters one or more search terms that the search engine uses to generate a listing of information, such as web pages, that the searcher is then able to access and utilize. The information resulting from the search is commonly identified as a result of an association that is established between the information and one or more of the search terms entered by the user. Different search engines use different techniques to associate information with search terms and to identify related information. These search engines also use different techniques to provide the identified information to the user. Accordingly, the likelihood of information being found as a result of a search varies depending upon the search engine used to perform the search.

This uncertainty is of particular concern to web page operators that make information available on the World Wide Web. In this setting, there are often several web page operators or advertisers that are competing for the same group of potential views or customers. Accordingly, a web page's ability to be identified as the result of a search is often important to the success of a web page. Therefore, web page operators often seek to increase the likelihood that their web page will be seen as the result of a search.

One type of search engine that provides web page operators with a more predictable method of being seen as the result of a search is a "pay for performance" arrangement where web pages are displayed based at least in part upon a monetary sum that the advertiser or web page operator has agreed to pay to the search engine operator. The web page operator agrees to pay an amount of money, commonly referred to as the bid amount, in exchange for a particular position in a set of search results that is generated in response to a user's input of a search term. A higher bid amount will result in a more prominent placement in a set of search results. Thus, a web page operator may attempt to place high bids on one or more search terms to increase the likelihood that their web page will be seen as a result of a search for that term. However, there are many similar search terms, and it is difficult for a web page operator to bid on every potentially relevant search term. Likewise, it is unlikely that a bid will be made on every search term. Accordingly, a search engine operator may not receive any revenue from searches performed using certain search terms for which there are no bids.

In addition, because the number of existing web pages is ever increasing, it is becoming more difficult for a user to find relevant search results. The difficulty of obtaining relevant search results is further increased because of the search engine's dependency on the search terms entered by the user. The search results that a user receives are directly dependent upon the search terms that the user enters. The entry of one search term may not result in relevant search results, while the entry of only a slightly different search term can result in relevant search results. Accordingly, the selection of search terms is often an important part of the search process. It would be of benefit to both the searcher and the advertisers to recommend related searches for

the searcher to provide to the search engine. However, current search engines do not enable a search engine operator to provide related search terms, such as those that will produce relevant search results, to a user. A system that overcomes these deficiencies is needed.

SUMMARY

By way of introduction only, in accordance with one embodiment of the invention, a search request is received from a searcher and used to perform a search on a pay for performance database. In the pay for performance database there are stored search listings including web page locators and bid amounts to be paid by the operator of the listed web page. The search using the pay for performance database produces search results which are presented to the searcher. The search request is also used to perform a search on a related search database. The related search database has been formed at least in part using contents of the pay for performance database. The search of the related search database produces a list of related searches which are presented to the searcher.

In accordance with a second embodiment, a related search database is created using a pay for performance database. All text from all web pages referenced by the pay for performance database is stored and used to create an inverted index. Additional indexes are used to improve the relevancy and spread of related search results obtained using the database.

The foregoing discussion of illustrative embodiments of the invention has been provided only by way of introduction. Nothing in this section should be taken as a limitation on the following claims, which define the scope of the invention.

BRIEF DESCRIPTION OF SEVERAL VIEWS OF THE DRAWINGS

FIG. 1 is a block diagram illustrating a database search system in conjunction with a computer network;

FIG. 2 is a flow diagram illustrating a method for operating the database search system of FIG. 1;

FIG. 3 is a flow diagram illustrating a method for operating the database search system of FIG. 1;

FIG. 4 is a flow diagram illustrating in more detail a portion of the method shown in FIG. 2;

FIG. 5 is a flow diagram illustrating in more detail a portion of the method shown in FIG. 2;

FIG. 6 is a flow diagram illustrating a method for forming a related searches database; and

FIG. 7 is a flow diagram illustrating a method for removing similar page information from a database.

DETAILED DESCRIPTION OF THE PRESENTLY PREFERRED EMBODIMENTS

Referring now to the drawing, FIG. 1 is a block diagram of a database search system 100 shown in conjunction with a computer network 102.

The database search system 100 includes a pay for performance database 104, a related searches database 106, a search engine web server 108, a related searches web server 110 and a search engine web page 114. The servers 104, 106, 108 may be accessed over the network 102 by an advertiser web server 120 or a client computer 122.

The network 102 in the illustrated embodiment is the Internet and provides data communication according to appropriate standards, such as Internet Protocol. In other embodiments, other network systems may be used alone or in conjunction with the Internet. Communication in the network 102 is preferably according to Internet Protocol or similar data communication standard. Other data communications standards may be used as well to ensure reliable communication of data.

The database search system 100 is configured as part of a client and server architecture. In the context of a computer network such as the Internet, a client is a process such as a program, task or application that requests a service which is provided by another process such as a program, task or application that requests a

service which is provided by another process, known as a server program. The client process uses the requested service without having to know any working details about the other server program or the server itself. In networked systems, a client process usually runs on a computer that accesses shared network resources provided by another computer running a corresponding server process. A server is typically a remote computer system that is accessible over a communications medium such as a network. The server acts as an information provider for a computer network. Thus, the system 100 operates as a server for access by the clients such as client computer 122 and the advertiser web server 120.

The client computers 122 can be conventional personal computers, workstations or computer systems of any size. Each client computer 112 typically includes one or more processors, memory, input and output devices and a network interface such as a modem. The advertiser web server 120, the search engine web server 108, the related searches web server 110 and the account management web server 112 can be similarly configured. However, the advertiser web server 120, the search engine web server 108, the related searches web server 110 and the account management web server 112 may each include many computers connected by a separate private network.

The client computer 112 executes a World Wide Web ("web") browser program 124. Examples of such a program are Navigator, available from Netscape Communications Corporation and Internet Explorer, available from Microsoft Corporation. The browser program 124 is used by a user to enter addresses of specific web pages to be retrieved. These addresses are referred to as Uniform Resource Locators (URLs). In addition, once a page has been retrieved, the browser program 124 can provide access to other pages or records when the user clicks on hyperlinks to other web pages contained in the web page. Such hyperlinks provide an automated way for the user to enter the URL of another page and to retrieve that page. The pages can be data records including as content plain textual information or more complex digitally encoded multimedia content such as software programs, graphics, audio data, video data and so forth.

Client computers 122 communicate through the network 102 with various network information providers. These information providers include the advertiser web server 120, the account management server 112, the search engine server 108, and the related searches web server 110. Preferably, communication functionality is provided by HyperText Transfer Protocol (HTTP), although other communication protocols such as FTP, SNMP, Telnet and a number of other protocols known in the art may be used. Preferably, search engine server 108, related searches server 110 and account management server 112, along with advertiser servers 120 are located on the worldwide web. U.S. Patent Application Number 09/322,627, filed May 29, 1999 and entitled "System and Method for Influencing a Position on a Search Result List Generated by a Computer Network Search Engine," and U.S. Patent Application No. 09/494,818, filed January 31, 2000 and entitled "Method and System for Generating a Set of Search Terms," are commonly assigned to the assignee of the present application and are incorporated herein by reference. These applications disclose additional aspects of search engine systems.

The account management web server 112 in the illustrated embodiment includes a computer storage medium such as a disc system and a processing system. A database is stored on the storage medium and contains advertiser account information. Conventional browser programs 124, running on client computers 122, may be used to access advertiser account information stored on the account management server 112.

The search engine web server 108 permits network users, upon navigating to the search engine web server URL or sites on other web servers capable of submitting queries to the search engine web server 108 through a browser program 124, to type keyword queries to identify pages of interest among the millions of pages available on web pages. In one embodiment of the present invention, the search engine web server 108 generates a search result list that includes, at least in part, relevant entries obtained from and formatted by the results of the bidding process conducted by the account management server 112. The search engine web server 108 generates a list of HyperText links to documents that contain

information relevant to search terms entered by the user at a client computer 122. The search engine web server transmits this list, in the form of a web page 114 to the network user, where it is displayed on the browser 124 running on the client computer 122. One embodiment of the search engine web server may be found by navigating to the web page at URL http://www.goto.com/.

Search engine web server 108 is connected to the network 102. In one embodiment of the present invention, search engine web server 108 includes a pay for performance database including a plurality of search listings. The database 104 contains and ordered collection of search listing records used to generate search results in response to user queries. Each search listing record contains the URL of an associated web page or document, a title, descriptive text and a bid amount. In addition, search engine web server 108 may also be connected to the account management server 112. The account management server 112 may also be connected to the network 102.

In addition, in the illustrated embodiment of FIG. 1, the database system 100 further includes a related searches web server 110 and an associated related searches database 106. The related searches web server 110 and data base 106 operate to provide suggested, related searches for presentation to a searcher along with search results in response to his query. Users conducting searches for information using a search engine web server such as the server 108 often perform searches which are inappropriately focused as compared to the index data of the web site search engine. Users may use search terms which are either to vague and generalized, such a "music," or too specific and focused, such as "hot jazz from New Orleans during the early 1950s." Some users require assistance to refine their query to better obtain useful information from the search engine. The related searches web server 110 provides the user with query suggestions better suited to the abilities of the pay for performance database 104.

In the illustrated embodiment, the pay for performance database 104 is established in conjunction with advertisers who operate web servers such as advertiser web server 120. Advertiser web pages 121 are displayed on the advertiser web server 120. An advertiser or web site promoter may, through an

account residing on the account management server 112, participate in a competitive bidding process with other advertisers. An advertiser may bid on any number of search terms relevant to the content of the advertisers web site.

The bids submitted by the web site promoters are used to control presentation of search results to a searcher using client computer 122. Higher bids receive more advantageous placement on a search result list generated by the search engine web server 108 when a search using the search term bid on by the advertiser is executed. In one embodiment, the amount bid by an advertiser comprises a money amount that is deducted from the account of the advertiser each time the advertiser web site is accessed via a hyperlink on the search result list page. A searcher clicks on the hyperlink with a computer input device such as a mouse to initiate a retrieval request to retrieve the information associated with the advertiser's hyperlink. Preferably, each access or click on a search result list hyperlink is redirected to the search engine web server 108 to associate the click with the account identifier for an advertiser. This redirection action, which is not apparent to the searcher, will access account information coded into the search result page before accessing the advertiser's URL using the search result list hyperlink clicked on by the searcher. In the illustrated embodiment, the advertiser's web site description and hyperlink on the search result list page is accompanied by an indication that the advertiser's listing is a paid listing. Each paid listing displays an amount corresponding to a price per click paid by the advertiser for each of referral to the advertiser site through this search result list.

The searcher may click on HyperText links associated with each listing in that search result page to access the corresponding web pages. The HyperText links may access web pages anywhere on the Internet, and include paid listings to advertiser web pages 121 located on the advertiser web server 120. In one embodiment of the present invention, the search result list also includes non-paid listings that are not priced as a result of advertiser's bids and are generated by a conventional search engine, such as the Inktomi, Lycos, or Yahoo! search engines. The non-paid HyperText links may also include links manually indexed into the

pay for performance database 104 by an editorial team. Preferably, the non-paid listings follow the paid advertiser listings on the search results page.

Related searches web server 110 receives the search request from the searcher at client computer 122 as entered using the search engine web page 114. In the related searches database 106, which includes related search listings generated from the pay for performance database 104, the related searches web server 110 identifies related search listings relevant to the search request. In conjunction with the search engine web server 108, the related searches web server 110 returns a search result list to the searcher including the identified search listings located in the pay for performance database and one or more identified related search listings located in the related searches database 106. Operation of the related searches web server 110 in conjunction with the related searches database 106 will be described below in conjunction with FIGS. 2-5. The formation of the related searches database 106 will be described below in conjunction with FIG. 6.

FIG. 2 is a flow diagram illustrating a method for operating the database search system 100 of FIG. 1. The method begins at block 200. Java source code for implementing the method of FIG. 2 and other method steps described herein is included as an appendix.

At block 202, a search request is received. The search request may be received in any suitable manner. It is envisioned that a search request will originate with a searcher using a client computer to access the search engine web page of the database system implementing the method illustrated in FIG. 2. A search request may be typed in as input text in a hyperlink click to initiate the search request and search process.

After block 202, two parallel processes are initiated. At block 204, the search engine web server of the database search system identifies matching search listings in the pay for performance database of the system. In addition, the search engine web server may further identify non-paid search listings.

Similarly, at block 206, a related searches web server initiates a search to identify matching related search listings in the related search database. By

matching search listings, it is meant that the respective search engine identifies search listings contained in the respective database which generate a match with the search request. A match may be generated if an exact, letter for letter textual match occurs between a bid on keyword and a search term. In other embodiments, a match may be generated if a bidded keyword has a predetermined relationship with a search term. For example, the predetermined relationship may include matching the root of a word which has been stripped of suffixes; in a multiple word query, matching several but not all of the words; or locating the multiple words of the query with a predetermined number of words of proximity.

After the search results have been located, the search results from the pay for performance database are combined with search results from the related search database, block 208. At block 210, a search result list is returned to the searcher, for example by displaying identified search listings on the search engine web page and conveying the web page data over the network to the client computer. The search results and related search results may be displayed in any convenient fashion.

An example of a search result list display used in one embodiment of the present invention is shown in FIG. 3, which is a display of the first several entries resulting from the search for the term "CD burners." The exemplary display of FIG. 3 shows a portion of a search result list including a plurality of entries 310a, 310b, 310c, 310d, 310e, 310f, 310g, 310h, 310i, a listing 312 of other search categories and a related searches listing 314.

As shown in FIG. 3, a single entry, such as entry 310a in the search result list consists of a description 320 of the web site, preferably comprising a title and a short textual description, and a hyperlink 330 which, when clicked by a searcher, directs the searcher browser to the URL where the described web site is located. The URL 340 may also be displayed in the search result list entry 310a, as shown in FIG. 3. The "click through" of a search result item occurs when the remote searcher viewing the search result item display 310 of FIG. 3 selects or clicks on the hyperlink 330 of the search result item display 310.

Search result list entries 310a-310h may also show the rank value 360a, 360b, 360c, 360d, 360e, 360f, 360g, 360h, 360i of the advertisers' search listing. The rank value 360a-360i is an ordinal value, preferably a number, generated and assigned to the search listing by the processing system of the search engine web server. Preferably, the rank value 360a-360i is assigned in a process, implemented in software, that establishes an association between the bid amount, the rank, and the search term of a search listing. The process gathers all search listings that match a particular search term, sorts the search listings in order from highest to lowest bid amount, and assigns a rank value to each search listing in order. The highest bid amount receives the highest rank value, the next highest bid amount receives the next highest rank value, proceeding to the lowest bid amount, which receives the lowest rank value. The correlation between rank value and bid amount is illustrated in FIG. 3, where each of the paid search list entries 310a-310h display the advertiser's bid amount 350a, 350b, 350c, 350d, 350e, 350f, 350g, 350h, 350i for that entry. If two search listings having the same search term also have the same bid amount, the bid that was received earlier in time will be assigned the higher rank of value.

The search result list of FIG. 3 does not include unpaid listings. In the preferred embodiment, unpaid listings do not display a bid amount and are displayed following the lowest ranked paid listing. Unpaid listings are generated by a search engine utilizing object distributed database and text searching algorithms as known in the art. An example of such a search engine is the search engine operated by Inktomi Corporation. The original search query entered by the remote searcher is used to generate unpaid listings through the conventional search engine.

The listing 312 of other search categories shows other possible categories for searching that may be related to the searcher's input search term 316. The other search categories are selected for display by identifying a group such as computer hardware containing the input search term 316. Categories with in the group are then displayed as hyperlinks which may be clicked through by the

searcher for additional searches. This enhances the user's convenience in cases where the user's input search did not turn up suitable search results.

The related searches listing 314 displays six entries 318 of related searches determined using the related searches database as described herein. In other embodiments, other numbers of related search entries may be show. In addition, a link 320 labeled "more" allows the user to display additional related search entries. In the illustrated embodiment, the displayed entries 318 are the top six most relevant and most bidded-on terms in the related searches database.

Referring now to FIG. 4, the act of identifying matching related search listings in a related search database (act 206, FIG. 2) in one embodiment comprises the following acts. At block 400, an inverted index containing all data from all web pages contained in the pay for performance database of the database search system is searched. The inverted index is stored in the related searches database. In an inverted index, a single index entry is used to reference many database records. Searching for multiple matches per index entry is generally faster when using inverted indexes, since each index entry may reference many database records. The inverted index lists the words which can be searched in, for example, alphabetical order and accompanying each word are pointers which identify the particular documents which contain the word as well as the locations within each document at which the word occurs. To perform a search, instead of searching through the documents in word order, the computer locates the pointers for the particular words identified in a search query and processes them. The computer identifies the documents which have the required order and proximity relationship for the search query terms.

At block 402, meta-information is also searched for the received search term. Meta-information is abstracted, once-removed information about the collected data itself and forms a description of the data. Meta-information is derived information and relational information. Meta-information for a listing describes the relation of the listing to other listings, and meta-information for a listing describes the relation of the advertisers sponsoring a listing to other advertisers.

Meta-information is obtained using a script of command to analyze the pay for performance data base and determine information and relationships present in the data. The meta-information is collected for each row of data in the database and attached to that row. In one embodiment, the script is run one time as a batch process after the data is collected in the database. In other embodiments, the script is periodically re-run to update the meta-information.

Meta-information about the web pages and key words contained in the pay for performance database includes information such as the frequency of occurrence of similar key words among different web site domains and the number of different key words associated with a single web site. The meta-information may further include fielded advertiser data which is the information contained in each search listing provided by web site promoters who have bid upon search terms in the pay for performance database; advertiser identification information; web site themes, such as gambling or adult content;, and derived themes. Preferably, the meta-information is combined in a common inverted index with the stored web page data searched at block 400.

The result of the searches of block 400 and block 402 is a listing of rows of the inverted index or indexes containing the searched information. Each row contains the information associated with a search listing of the pay for performance database along with all the text of the web page associated with the search listing. In the illustrated embodiment, the search listing includes the advertiser's search terms, the URL of the web page, a title and descriptive text.

At block 404, the returned related search results are sorted by relevancy. Any suitable sorting routine may be used. A preferred process of sorting the search results by relevancy, block 404, is illustrated in greater detail in FIG. 5.

At block 406, the six most relevant related search results are selected. It is to be noted that any suitable number of search results may be provided. The choice of providing six related searches as suggestions to a searcher is arbitrary. After block 406, control proceeds at block 208, FIG. 2.

FIG. 5 is a flow diagram illustrating a method for sorting by relevancy search results obtained from a related searches database, corresponding to

block 404 of FIG. 4. In the embodiment illustrated in FIG. 5, a relevancy value is maintained for each returned listing. The relevancy value is adjusted according to specific relevancy factors, some of which are defined in FIG. 5. Other relevancy factors may be used as well. After adjusting the relevancy value, a final sorting occurs and the highest valued listings are returned.

At block 500, the relevancy value for individual records located during the search (block 400, block 402, FIG. 4) are increased according to the frequency of occurrence of a queried search term in each respective record. For example, if the queried search term occurs frequently in the text associated with the search listing, the relevancy of that listing is increased. If the queried search term occurs rarely or not at all in the listing, the relevancy value of that list is not increased or is decreased.

At block 502, it is determined if there are multiple search terms in the search queries submitted by the searcher. If not, control proceeds to block 506. If there are multiple search terms, at block 504, the relevancy of individual search results is increased according to proximity of the searched terms in a located record. Thus, if two search terms are immediately proximate, the relevancy score value for the record may be substantially increased, suggesting that the identified search listing is highly relevant to the search query submitted by the searcher. On the other hand, if the two search terms occur, for example, in the same sentence but not in close proximity, the relevancy of the record may be slightly increased to indicate the lesser relevancy suggested by the reduced proximity of the search terms.

At block 506, it is determined if the located record contains a bidded search term. Search terms are bidded on by advertisers, the bids being used for display of search results by the search engine web server using the pay for performance database. If the search result does include a bided on search term, the relevancy of the record is adjusted, block 508. If the query does not include one or more bided on search terms, control proceeds to block 510.

At block 510, it is determined if there are search terms in the description of the search listing. As illustrated in FIG. 3, each such listing includes a textural

description of the contents of the web site associated with the search listing. If the search terms are not included in the description, control proceeds to block 514. If the search terms are included in the description, at block 512, the relevancy of the located record is adjusted accordingly.

At block 514, it is determined if the search terms are located in the title of the search listing. As illustrated in FIG. 3, each search listing includes a title 360. If the search terms are included in the title of a record, the relevancy of the record is adjusted accordingly, block 516. If the search terms are not included in the title, control proceeds to block 518.

At block 518, it is determined if the search terms are included in the metatags of the search listing. Metatags are textural information included in a web site which is not displayed for user use. However, the search listing contained in the pay-for-performance database includes the metatags for searching and other purposes. If, at block 518, the search terms are not included in the search listing, control proceeds to block 522. On the other hand, if the search terms are included in one or more metatags of the search listing, at block 520 the relevancy of the record is adjusted accordingly.

At block 522 it is determined if the user's search terms are included in the text of the bided web page. If not, control proceeds to block 406, FIG. 4. However, if the search terms are included in the web page text, at block 524 the relevancy of the search listing record is adjusted accordingly.

Following the steps illustrated in FIG. 5, one or more and preferably six most relevant related search listings are returned and presented to the searcher along with the search results from the pay-for-performance database.

FIG. 6 illustrates a method for forming a related searches database for use in the database search system of FIG. 1. The method begins at block 600.

At block 602, all text for all web pages in the pay-for-performance database is fetched. This includes metatags and other non-displayed textual information contained in the web page referenced by a URL contained in the pay for performance database. At block 604, text from similar pages is omitted. This reduces the amount of data which must be processed to form the related searches

database. One embodiment of a method for performing this act will be described below in conjunction with FIG. 7. In addition, this greatly increases the speed at which the related searches database may be produced. At block 606, the text is stored in the related searches database.

At block 608, an inverted index is created, indexing the search listing data stored at block 606 along with the text fetched at block 602. The resulting inverting index includes a plurality of rows of data, each row including a key word along with all text from the database associated with that key word.

One illustrative example of a configuration for the contents of the related search database follows. Each row of the database includes the following elements:

canon_cnt	integer	# Number of different search listings bidded on this related result
advertiser_cnt	integer	# Number of different advertisers bidding on this related result
related_result	varchar(50)	# related result (bidded search term), canonicalized and depluralized
raw_search_text	varchar(50)	# original raw bidded search term
advertiser_ids	varchar(4096)	# explicit list of all advertisers bidding on this related result
words	varchar(65536+)	# full text of all web pages crawled, including hand-coded descriptions
theme	varchar(50)	
directory_taxonomy	varchar(200)	

The count canon_cnt differs from the count advertiser_cnt because many different web pages in the same domain could be bidded against the same bidded search term, or many different advertisers may bid on only 1 search term. Special themed keys are embedded into the database with 'flags' inserted in the advertiser_cnt field. If 'advertiser_cnt == 999999999', the query being presented is an adult-oriented query. In this implementation, an optional enhancement is to disable related results in this case. The counts canon_cnt and advertiser_cnt are the current derived-data fields. Additional fields such as theme and

directory_taxonomy_category can optionally be added to give even more enhanced relevance to related results matches, though they are not used in the illustrated embodiment.

In one embodiment, the inverted index which is queried against to obtain the related results is created with the following Java command:

SQL> Create metamorph inverted index mm_index02 on line_ad02(words);

This is the vendor-specific method (using the Texis relational database management system provide by Thunderstone- EPI, Inc.) for creating a free-text search index (mm_index02) on a document (here contained in a database column (words)) which will be searched (from RelatedSearcherCore.java) by the Texis Thunderstone SQL command:

```
"SELECT"
+ "$rank, "          //Num  getRow() arg position 0
+ "canon_cnt, "      //Int  getRow() arg position 1
+ "raw_search_text, "  //Stri getRow() arg position 2
+ "cannon_search_text, "     //Stri getRow() arg position 3
+ "advertiser_ids, "  //Stri getRow() arg position 4
+ "advertiser_cnt "   //Int  getRow() arg position 5
+ "FROM line_ad02 "
+ "WHERE words "
+ "LIKEP $query ORDER BY 1 desc, advertiser_cnt desc;";
```

The $rank is a vendor-supplied virtual data field which programmatically contains the "relevancy" of the search result, based on the frequency of occurrence of the queried phrase ($query) in the "words" field, the proximity of the queried phrase elements to each other within the indexed words field, and the word order (if > 1 query phrase word) as compared to the ordering of words within the "words" field.

The "rank" is vendor-specific, and derived by various differing algorithms by different Free Text Search Engine suppliers, though is similar enough in practice that any vendor's Free Text Search Engine works to implement the Related Searches Functionality.

The "ORDER BY 1 desc[ending], advertiser_cnt desc[ending]" controls ranking the results of the query by relevance primarily (field "1" == $rank), and secondarily by the derived field "advertiser_cnt", which is the count of advertisers

bidding on this particular related_search_result. Thus, "relevance" is the primary selection criteria, and "popularity" is the secondary selection criteria.

At block 610, additional indexes are created and stored with the inverted index created at block 608. The additional indexes are created using key information associated with each search listing. The key information includes, for example, fielded advertiser data such as an advertiser's identification and derived themes such as gambling and so forth. The method then ends at block 612.

FIG. 7 is a flow diagram illustrating a method for removing similar page information from a database. The method in the illustrated implementation follows performance of act 602 of FIG. 6.

At block 702, the pay for performance database (also referred to as a bidded search listing data base) is examined for URL data and all URLs are extracted from the database and formed into a list. The list is sorted and any exact duplicates are removed, block 704

At block 706, a URL in the list is selected and it is determined if the selected URL bears similarity to a preceding URL in the list. Similarity may be determined by any suitable method, such as a number of identical characters or fields within the URL or a percentage of identical characters, or a common root or string or field.

At block 708, if the selected URL is similar to the preceding URL, the selected URL is added to a list of candidate duplicate URLs. At block 710, a predetermined number of each potentially duplicate URL are crawled. In the illustrated embodiment, the predetermined number is the first two potentially duplicate URLs. Crawling is preferably accomplished using a program code referred to as a crawler. A crawler is a program that visits Web sites and reads their pages and other information. Such programs are will known and are also known as a "spiders" or "bots." Entire sites or specific pages can be selectively visited and indexed by a crawler. In alternative embodiments, subsets of each site referenced by a URL, rather than an entire site, may be crawled and compared for similarity.

At block 712, the data returned by the crawler is examined. The data may be referred to as the body of the URL and includes data from the site identified by the URL and all accessible pages of the site. It is determined if the data including text and other information contained in the body of the URL is sufficiently similar to the data contained in the body of the previous URL. Again, similarity may be determined by any suitable method, such as a statistical comparison of the textual content of each page. If there is sufficient similarity, control proceeds to block 714 and it is assumed that the URL is the same as the previous URL. The body of text and other information is assigned to the rest of the similar URLs.

If, at block 706, it was determined that the selected URL was not similar to the preceding URL, or if at block 712 it was determined that the body of the URL was not similar enough to the body of the previous URL, control proceeds to block 718. At block 718, the URL is added to a list of URLs to be crawled. At block 720, all URLs on the list are crawled to retrieve and store information contained at the sites indicated by each URL.

At block 716, the information from each crawled URL is loaded into the related searches database (also referred to as the free text database). The information is joined with search listing data already included in the related searches database. Thus, the method steps illustrated in FIG. 7 reduce the total amount of data contained in the related searches database by reducing the number of URLs that are crawled and stored. Duplicate URLs are eliminated from the process and near-duplicate URLs are checked for similarity of content. The result is reduced storage requirements for the resulting database and faster, more efficient searching on the database. This enhances user convenience by improving performance.

From the foregoing, it can be seen that the present invention provides an improved method and apparatus for producing related searches for presentation to a searcher searching in a pay for performance database. Related searches are performed in a related searches database which has been formed using the pay for performance database. The search results from the related searcher's database are ordered by relevancy for presentation to the user. Thus, if a user's initial search

was too narrow or too broad, the user has available related searches which may be used to produce more usable results. In addition, the related searches have been produced using search listings referenced by bidded search terms. This provides a benefit to advertisers who pay for advertising in the database search system. This increases the likelihood that an advertiser's web site will be visited by a searcher using the database system.

While a particular embodiment of the present invention has been shown and described, modifications may be made. It is therefore intended in the appended claims to cover all such changes and modifications which fall within the true spirit and scope of the invention.

CLAIMS

1. A method of generating a search result list, the method comprising:

receiving a search request from a searcher; in a pay for performance database including a plurality of search listings, identifying search listings generating a match with the search request;

in a related search database including related search listing generated from the pay for performance database, identifying related search listings relevant to the search request; and

returning a search result list to the searcher including the identified search listings and one or more of the identified related search listings.

2. The method of claim 1 wherein identifying related search listings comprises:

searching an inverted index of the pay for performance database; and

searching an index of meta-information based on the pay for performance database.

3. The method of claim 1 further comprising:

sorting the identified related search listings by relevancy to the search request;

selecting a predetermined number of the identified related search listings as most relevant related search listings; and

returning the most relevant related search listings in the search result list.

4. The method of claim 3 wherein sorting comprises:

selecting the identified related search listings according to frequency of occurrence of a queried term from the search request in the related search listings.

5. The method of claim 3 wherein sorting comprises:

selecting the identified related search listings according to proximity of one or more queried terms from the search request in the related search listings.

6. The method of claim 3 wherein sorting comprises:

weighting the related search listings according to predetermined weighting criteria; and

selecting the identified related search listings according to the weighting of the related search listings.

7. The method of claim 6 wherein weighting the related search listings comprises:

increasing relative weighting of a related search listing which includes one or more bided search terms identified by an advertiser.

8. The method of claim 6 wherein weighting the related search listings comprises:

increasing relative weighting of a related search listing which is contained in a description of a search listing identified by an advertiser.

9. The method of claim 6 wherein weighting the related search listings comprises:

increasing relative weighting of a related search listing which is contained in a title of a search listing identified by an advertiser.

10. The method of claim 6 wherein weighting the related search listings comprises:

increasing relative weighting of a related search listing which is contained in metatag keywords of a web page maintained by an advertiser.

11. The method of claim 6 wherein weighting the related search listings comprises:

increasing relative weighting of a related search listing which is contained in text data of a web page maintained by an advertiser.

12. The method of claim 3 wherein sorting comprises:

ranking the related search listings according to spread of the related search listings; and

selecting the identified related search listings according to the ranking of the related search listings.

13. The method of claim 12 wherein ranking comprises:

identifying key information contained in the related search listings; and

increasing ranking of a related search listing according to presence of the key information in the related search listing.

14. The method of claim 13 wherein identifying key information comprises:

detecting fielded advertiser data in the related search listing; and

detecting crawled data in the related search listing.

15. A system comprising:

a pay for performance database;

a related search database formed at least in part using the pay for performance database; and

a server coupled with the pay for performance database and the related search database, the server operative to select a first set of search results from the pay for performance database and a second set of search results from the related search database in response to a search request from a searcher.

16. The system of claim 15 wherein the pay for performance database comprises:

a plurality of search listings, each search listing including

a search term,

a bid amount, and

a Uniform Resource Locator corresponding to an address of a document on a network server remote from the system.

17. The system of claim 16 wherein the related search database comprises:

a plurality of related search listings, each related search listing including

a keyword associated with one document of the pay for performance database, and

text of the one document.

18. The system of claim 17 wherein each search listing of the plurality of search listings further comprises:

descriptive text describing the one document,

a title, and

metatags associated with the document.

19. The system of claim 18 wherein each search listing comprises:

the descriptive text associated with the one document;

the title associated with the one document; and

the metatags associated with the one document.

20. A method for forming a related searches database for identifying related searches in response to a search request to a pay for performance database including a plurality of search listings, the method comprising:

storing as a related search database entry text from each web page referenced by a search listing of the pay for performance database;

creating an inverted index for the related search database entries; and

creating an index for key information associated with each search listing of the pay for performance database.

21. The method of claim 20 wherein storing comprises:

identifying similar web pages responsive to root path components and query arguments of Uniform Resource Locators for two or more web pages referenced by search listings of the pay for performance database;

rejecting for storage similar web pages.

22. The method of claim 21 wherein identifying similar web pages comprises:

identifying first key words of a first web page;

identifying second key words of a second web page; and

comparing the first key words and the second key words;

when the first key words and the second key words have a predetermined relationship, identifying the first web page and the second web page as similar web pages.

23. A method for searching data in a database including internet data from internet web sites, the method comprising:

forming a list of uniform resource locators (URLs) associated with internet web sites to be accessed;

removing duplicate URLs from the list;

if a URL on the list is similar to another URL on the list, crawling a predetermined number of potentially duplicate URLs;

comparing bodies of the URL on the list and the potentially duplicate URLs;

if the body of the URL on the list is similar to the body of the potentially duplicate URL,

suspending crawling of the potentially duplicate URLs, and

storing the body of the URL on the list in the database for subsequent search.

24. The method of claim 23 further comprising:

comparing a selected URL with other URLs on the list; and

determining the URL is similar to the other URL on the list when the URL has a predetermined text portion in common with the other URL on the list.

25. The method of claim 23 wherein comparing bodies of the URL on the list and the potentially duplicate URLs comprises:

comparing text from the URL on the list and text from one potentially duplicate URL; and

determining the URL on the list is similar to the one potentially duplicate URL when the text from the URL on the list and the text from the one potentially duplicate URL have a predetermined text portion in common.

ABSTRACT OF THE DISCLOSURE

A method of generating a search result list also provides related searches for use by a searcher. Search listings which generate a match with a search request submitted by the searcher are identified in a pay-for-performance database which includes a plurality of search listings. Related search listings contained in a related search database generated from the pay-for-performance database are identified as relevant to the search request. A search result list is returned to the searcher including the identified search listings and one or more of the identified search listings.

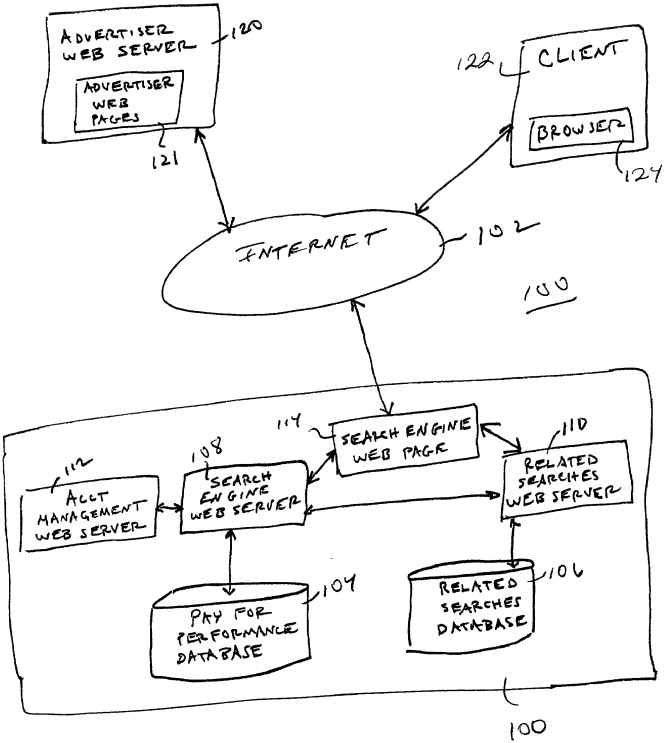

ADVERTISER WEB SERVER
120
ADVERTISER WEB PAGES
121
122
CLIENT
BROWSER
124
INTERNET
102
100
114
SEARCH ENGINE WEB PAGE
110
112
ACCT MANAGEMENT WEB SERVER
108
SEARCH ENGINE WEB SERVER
RELATED SEARCHES WEB SERVER
PAY FOR PERFORMANCE DATABASE
104
RELATED SEARCHES DATABASE
106
100

FIG. 1

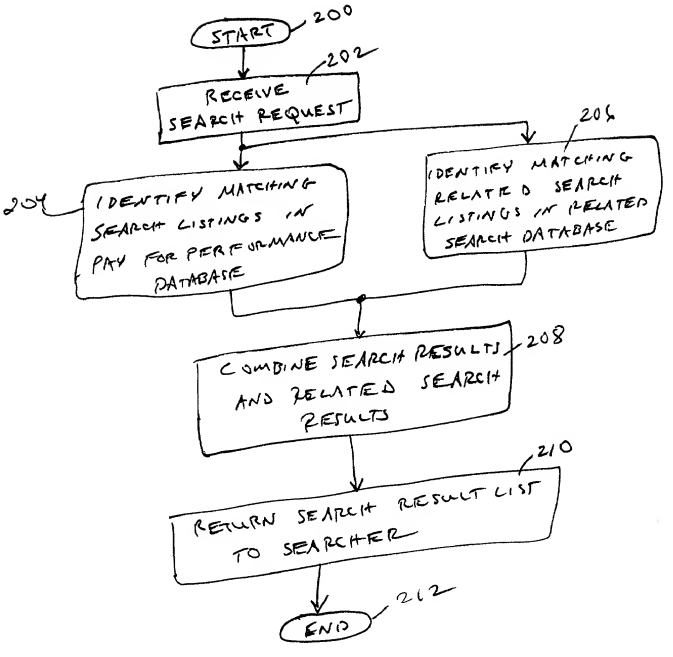

START
200
RECEIVE SEARCH REQUEST
202
IDENTIFY MATCHING SEARCH LISTINGS IN PAY FOR PERFORMANCE DATABASE
204
IDENTIFY MATCHING RELATED SEARCH LISTINGS IN RELATED SEARCH DATABASE
206
COMBINE SEARCH RESULTS AND RELATED SEARCH RESULTS
208
RETURN SEARCH RESULT LIST TO SEARCHER
210
END
212

FIG. 2

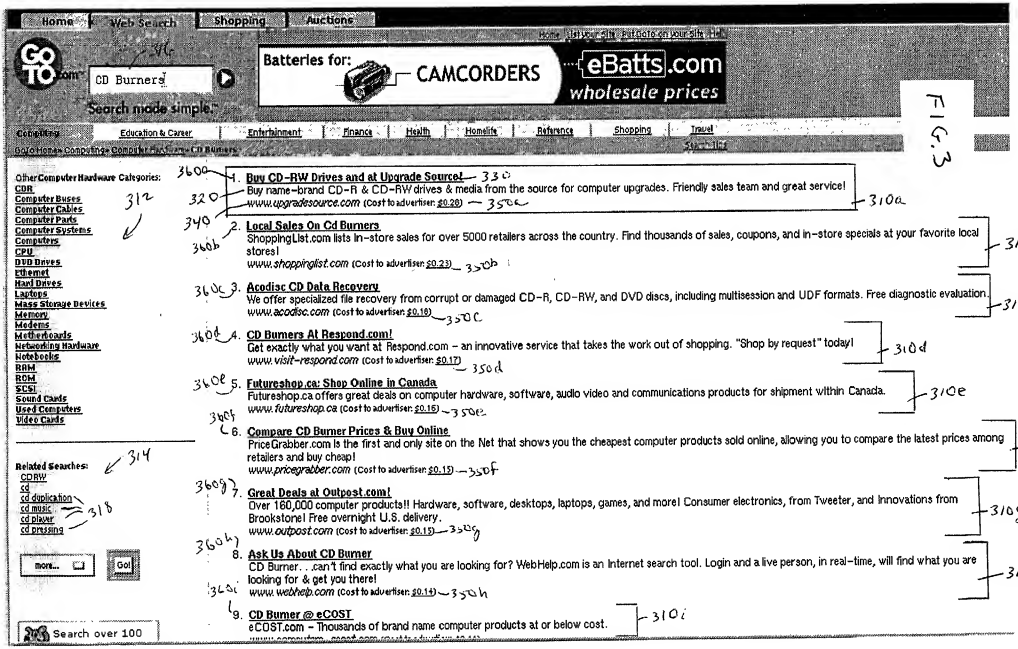

Home
Web Search
Shopping
Auctions
GO TO .com
CD Burners
Search made simple.
Batteries for:
CAMCORDERS
eBatts.com
wholesale prices
Computing
Education & Career
Entertainment
Finance
Health
Homelife
Reference
Shopping
Travel
Other Computer Hardware Categories:
CDR
Computer Buses
Computer Cables
Computer Parts
Computer Systems
Computers
CPU
DVD Drives
Ethernet
Hard Drives
Laptops
Mass Storage Devices
Memory
Modems
Motherboards
Networking Hardware
Notebooks
RAM
ROM
SCSI
Sound Cards
Used Computers
Video Cards
Related Searches:
CDRW
cd
cd duplication
cd music
cd player
cd pressing
more...
Go!
Search over 100
1. Buy CD-RW Drives and at Upgrade Source!
Buy name-brand CD-R & CD-RW drives & media from the source for computer upgrades. Friendly sales team and great service!
www.upgradesource.com (Cost to advertiser: $0.28)
2. Local Sales On Cd Burners
ShoppingList.com lists in-store sales for over 5000 retailers across the country. Find thousands of sales, coupons, and in-store specials at your favorite local stores!
www.shoppinglist.com (Cost to advertiser: $0.23)
3. Acodisc CD Data Recovery
We offer specialized file recovery from corrupt or damaged CD-R, CD-RW, and DVD discs, including multisession and UDF formats. Free diagnostic evaluation.
www.acodisc.com (Cost to advertiser: $0.18)
4. CD Burners At Respond.com!
Get exactly what you want at Respond.com - an innovative service that takes the work out of shopping. "Shop by request" today!
www.visit-respond.com (Cost to advertiser: $0.17)
5. Futureshop.ca: Shop Online in Canada
Futureshop.ca offers great deals on computer hardware, software, audio video and communications products for shipment within Canada.
www.futureshop.ca (Cost to advertiser: $0.15)
6. Compare CD Burner Prices & Buy Online
PriceGrabber.com is the first and only site on the Net that shows you the cheapest computer products sold online, allowing you to compare the latest prices among retailers and buy cheap!
www.pricegrabber.com (Cost to advertiser: $0.15)
7. Great Deals at Outpost.com!
Over 160,000 computer products!! Hardware, software, desktops, laptops, games, and more! Consumer electronics, from Tweeter, and Innovations from Brookstone! Free overnight U.S. delivery.
www.outpost.com (Cost to advertiser: $0.15)
8. Ask Us About CD Burner
CD Burner. . .can't find exactly what you are looking for? WebHelp.com is an Internet search tool. Login and a live person, in real-time, will find what you are looking for & get you there!
www.webhelp.com (Cost to advertiser: $0.14)
9. CD Burner @ eCOST
eCOST.com - Thousands of brand name computer products at or below cost.
360a
320
340
360b
360c
360d
360e
360f
360g
360h
360i
312
314
318
310a
310b
310c
310d
310e
310f
310g
310h
310i
330
350a
350b
350c
350d
350e
350f
350g
350h

FIG. 3

3/8

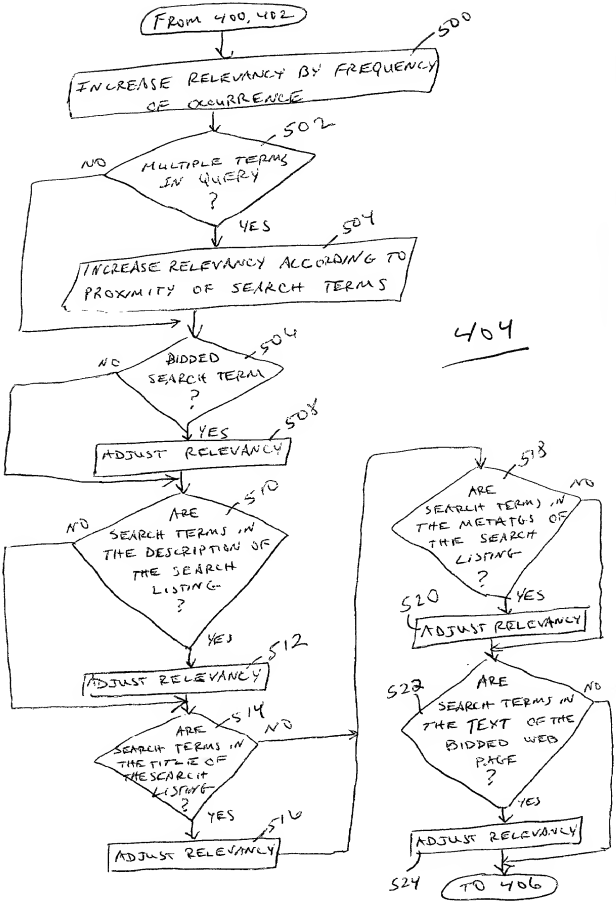
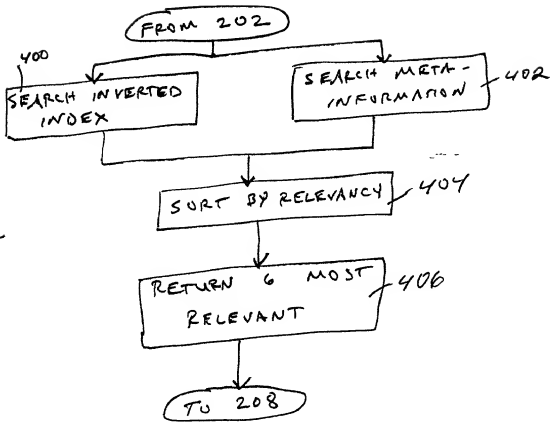

FROM 400, 402
500
INCREASE RELEVANCY BY FREQUENCY OF OCCURRENCE
502
MULTIPLE TERMS IN QUERY ?
NO
YES
504
INCREASE RELEVANCY ACCORDING TO PROXIMITY OF SEARCH TERMS
404
506
BIDDED SEARCH TERM ?
NO
YES
508
ADJUST RELEVANCY
510
ARE SEARCH TERMS IN THE DESCRIPTION OF THE SEARCH LISTING ?
NO
YES
512
ADJUST RELEVANCY
514
ARE SEARCH TERMS IN THE TITLE OF THE SEARCH LISTING ?
NO
YES
516
ADJUST RELEVANCY
518
ARE SEARCH TERMS IN THE METATAG OF THE SEARCH LISTING ?
NO
YES
520
ADJUST RELEVANCY
522
ARE SEARCH TERMS IN THE TEXT OF THE BIDDED WEB PAGE ?
NO
YES
524
ADJUST RELEVANCY
TO 406

FIG. 5



206

FIG. 4

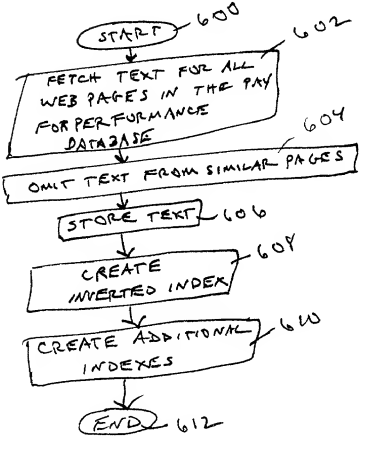
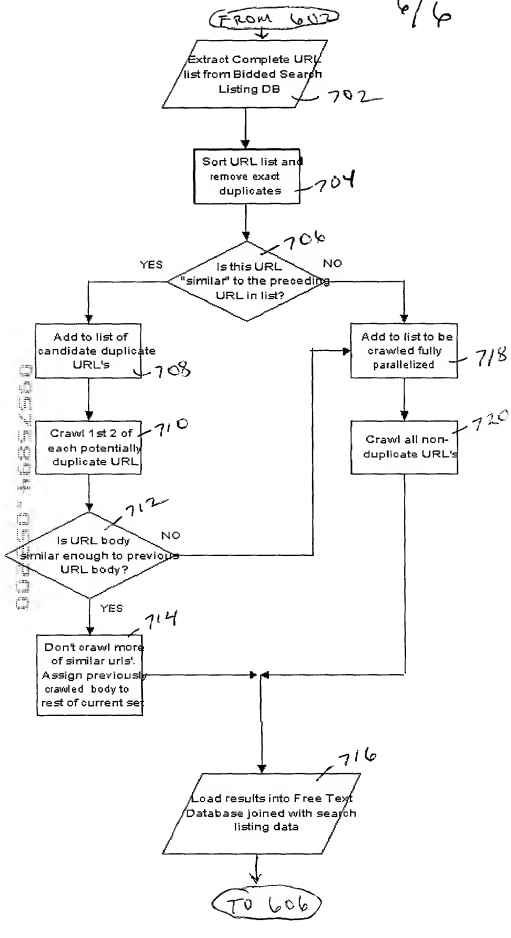


FIG. 6


FROM 602
Extract Complete URL list from Bidded Search Listing DB
702
Sort URL list and remove exact duplicates
704
706
Is this URL "similar" to the preceding URL in list?
YES
NO
Add to list of candidate duplicate URL's
708
Add to list to be crawled fully parallelized
718
Crawl 1st 2 of each potentially duplicate URL
710
Crawl all non-duplicate URL's
720
712
Is URL body similar enough to previous URL body?
NO
YES
714
Don't crawl more of similar urls'. Assign previously crawled body to rest of current set
716
Load results into Free Text Database joined with search listing data
TO 606

604

FIG. 7

Source Code Appendix

This is the core Java piece that actually performs the Free-text-search from the Free Text Search engine (Texis RDBMS from Thunderstone - EPI, Inc.), and post-processes the results, filtering on advertiser frequency of occurrence and theme ('adult'-ness).

```
package com.go2.search.related;

import java.util.Vector;
import java.util.Hashtable;
import java.util.StringTokenizer;
import java.rmi.RemoteException;

import com.go2.texis.*;


/**
*  @author Phil Rorex
*  @version
*/

class Callback implements TErrorMsgIF {
    private static int err115 = 0;
    public int getErr115() {
  return(err115);
    }
    public void ErrorMsgDelivery(String msg, int level, int
msgNumber) {
  switch (msgNumber){
  case 2: {
    System.out.println("FATAL: msg: " + msg + " level: " + level + "
msgNumber: " + msgNumber);
    System.exit(2);
  }
  case 100: break;
  case 115: err115++; break;
  default:
            System.out.println("UNUSUAL: msg: " + msg + " level: " +
level + " msgNumber: " + msgNumber);
  }
    }
}

/**
 * run as a stand-alone JVM, since the Free Text Searcher
 * being used is best connected with as a JNI-based C language
 * library interface API
 **/
```

```
public class RelatedSearcherCore implements Runnable {
  //cache an instance of Texis server and Query
  private static Server texis = null;
  private static Query texisQuery = null;
  private static Query texisPlurQuery = null;
  private static Query texisAdultQuery = null;
   private static long timeClock;

  //used to coordinate time-outs on extra long queries
  private static final Integer PRE_QUERY = new Integer(1);
  private static final Integer MID_QUERY = new Integer(2);
  private static final Integer POST_QUERY = new Integer(3);

   private static Integer semaphore = PRE_QUERY;

  // time out process
  Thread watchDog;

  // Thread starts out waiting on eternity
  // may never be used if Core() doesn't have timeout set for it
  long globalTimeOut = 0;

  //The magic adult flag
  //If a related search free-text search returns a row
  //which has this field set, it's automatically "themed"
  //as an adult-oriented related search
  //This particular "Magic data row" is pre-loaded with
  //all the "adult-oriented" terms which are typical
  //in this theme. Same should be done for CASINO_FLAG
  //CURRENT_NEWS_FLAG, and any other theme desired.
  private static final int ADULT_FLAG = 999999999;

  //How many pluralized tries of the query
  // used to search for singular and plural
  // version of up to [square root of] MAX_PLURAL_QRY
  // terms
  private static final int MAX_PLURAL_QRY = 4;

  //Limit Texis to this many rows
  //This is the initial # of pre-filtered free-text
  //searched rows coming back from the search engine
  private static final int MAX_ROWS = 60;

  //
  //controls the 'looseness' of the post-search filter
  //that filters out related searches based on the
  //derived-data element of (#-of-different-advertisers-
  //bidding on this related search term). Set to 0 is
  //this element means "how many times we can ignore seeing
  //the identical advertiser before we start ignoring
  //related searches bid on by him" 0 is strongest reject,
  //larger numbers reject less stringently (usu. not > than
  //1, if ratio of webpages:related searchterms is > than
  //about 10
  private static final int ADVERTISER_THRESHOLD = 0;

  //the SQL query used to talk to Texis (the FTS engine)
```

```
  private static final String TEXIS_SQL =
    "SELECT "
      + "$rank, "     //Num  getRow() arg position 0
      + "canon_cnt, "     //Int  getRow() arg position 1
      + "raw_search_text, "  //Stri getRow() arg position 2
      + "cannon_search_text, "  //Stri getRow() arg position 3
      + "advertiser_ids, "  //Stri getRow() arg position 4
      + "advertiser_cnt "  //Int  getRow() arg position 5
      + "FROM line_ad02 "
      + "WHERE words "
      + "LIKEP ? ORDER BY 1 desc, advertiser_cnt desc;";
      //+ "LIKEP ? ;";
      //+ "LIKEP ? ORDER BY advertiser_cnt desc;";

  private static final String TEXIS_PLUR_SQL =
    "SELECT plural "
      + "FROM plurals "
      + "WHERE singular "
      + "= ? ;";

  private static final String TEXIS_ADULT_SQL =
    "SELECT cannon_search_text "
      + "FROM adult "
      + "WHERE words "
      + "LIKE ? ;";

  private static Callback cb = new Callback();

public void init(String texisHome, long timeOut){
  globalTimeOut = timeOut;
  init(texisHome);
}

public void init(String texisHome){
/**
*   Instantiate Texis connection object and perform
*  Texis query initialization
*  Called one time to setup the Related Search query.
*  Must be called before findRelate is ever called.
*/
  //Perform Texis initialization and precache an instance
  //of Texis Server and Texis Query
  try {
                Texis texisRDBMS = new Texis();
    texis =  (Server) texisRDBMS.createServer(texisHome);

    Vector n = new Vector(200);
  //  Vector n = texis.getNoise();
    n.addElement("a");  n.addElement("about");
n.addElement("after");
    n.addElement("again");  n.addElement("ago"); n.addElement("all");
    n.addElement("almost");  n.addElement("also");
n.addElement("always");
    n.addElement("am");  n.addElement("an");  n.addElement("and");
    n.addElement("another");n.addElement("any");
n.addElement("anybody");
    n.addElement("anyhow");  n.addElement("anyone");
n.addElement("anything");
```

```
        n.addElement("anyway");  n.addElement("are");
    n.addElement("as");
        n.addElement("at");  n.addElement("away");  n.addElement("be");
        n.addElement("became");
    n.addElement("because");n.addElement("been");
        n.addElement("before");  n.addElement("being");
    n.addElement("but");
        n.addElement("by");  n.addElement("came");  n.addElement("can");
        n.addElement("cannot");  n.addElement("com");
    n.addElement("come");
        n.addElement("could");  n.addElement("de");  n.addElement("del");
        n.addElement("der");  n.addElement("did");  n.addElement("do");
        n.addElement("does");  n.addElement("doing");
    n.addElement("done");
        n.addElement("down");  n.addElement("each");
    n.addElement("else");
        n.addElement("even");  n.addElement("ever");
    n.addElement("every");
        n.addElement("everyone");  n.addElement("everything");
    n.addElement("for");
        n.addElement("from");  n.addElement("front");
    n.addElement("get");
        n.addElement("getting");n.addElement("go");
    n.addElement("goes");
        n.addElement("going");  n.addElement("gone");
    n.addElement("got");
        n.addElement("gotten");  n.addElement("had");
    n.addElement("has");
        n.addElement("have");  n.addElement("having");
    n.addElement("he");
        n.addElement("her");  n.addElement("here");  n.addElement("him");
        n.addElement("his");  n.addElement("how");  n.addElement("i");
        n.addElement("if");  n.addElement("in");  n.addElement("into");
        n.addElement("is");  n.addElement("isn't");  n.addElement("it");
        n.addElement("jpg");  n.addElement("just");
    n.addElement("last");
        n.addElement("least");  n.addElement("left");
    n.addElement("less");
        n.addElement("let");  n.addElement("like");
    n.addElement("make");
        n.addElement("many");  n.addElement("may");
    n.addElement("maybe");
        n.addElement("me");  n.addElement("mine");  n.addElement("more");
        n.addElement("most");  n.addElement("much");  n.addElement("my");
        n.addElement("myself");  n.addElement("net");
    n.addElement("never");
        n.addElement("no");  n.addElement("none");  n.addElement("not");
        n.addElement("now");  n.addElement("of");  n.addElement("off");
        n.addElement("on");  n.addElement("one");  n.addElement("onto");
        n.addElement("org");  n.addElement("our");
    n.addElement("ourselves");
        n.addElement("out");  n.addElement("over");  n.addElement("per");
        n.addElement("put");
    n.addElement("putting");n.addElement("same");
        n.addElement("saw");  n.addElement("see");  n.addElement("seen");
        n.addElement("shall");  n.addElement("she");
    n.addElement("should");
```

```
        n.addElement("so");  n.addElement("some");
    n.addElement("somebody");
        n.addElement("someone");n.addElement("something");
    n.addElement("stand");
        n.addElement("such");  n.addElement("sure");
    n.addElement("take");
        n.addElement("than");  n.addElement("that");
    n.addElement("the");
        n.addElement("their");  n.addElement("them");
    n.addElement("then");
        n.addElement("there");  n.addElement("these");
    n.addElement("they");
        n.addElement("this");  n.addElement("those");
    n.addElement("through");
        n.addElement("till");  n.addElement("to");  n.addElement("too");
        n.addElement("two");  n.addElement("unless");
    n.addElement("until");
        n.addElement("up");  n.addElement("upon");  n.addElement("us");
        n.addElement("very");  n.addElement("was");  n.addElement("we");
        n.addElement("went");  n.addElement("were");
    n.addElement("what");
        n.addElement("what's");  n.addElement("whatever");
    n.addElement("when");
        n.addElement("where");  n.addElement("whether");
    n.addElement("which");
        n.addElement("while");  n.addElement("who");
    n.addElement("whoever");
        n.addElement("whom");  n.addElement("whose");
    n.addElement("why");
        n.addElement("will");  n.addElement("with");
    n.addElement("within");
        n.addElement("without");n.addElement("won't");
    n.addElement("would");
        n.addElement("wouldn't");  n.addElement("www");
    n.addElement("yet");
        n.addElement("you");  n.addElement("your");
        texis.setNoise(n);

        texisQuery = (Query) texis.createQuery();
        texisPlurQuery = (Query) texis.createQuery();
        texisAdultQuery = (Query) texis.createQuery();

        /*
         * Query.api()'s affect ALL queries, not just ones set on
         */
        texisQuery.setlikeprows(MAX_ROWS);
        texisQuery.allinear(0);
        texisQuery.alpostproc(0);

        texisQuery.prepSQL(TEXIS_SQL);
        texisPlurQuery.prepSQL(TEXIS_PLUR_SQL);
        texisAdultQuery.prepSQL(TEXIS_ADULT_SQL);

            TErrorMsg.RegisterMsgDelivery(cb);

        watchDog = new Thread(this);
        watchDog.setPriority(Thread.NORM_PRIORITY + 1);
        watchDog.start();
```

```
  }catch(TException te) {
    te.printStackTrace();
    throw new RuntimeException(
      "Could not initialize Texis: Failed with: " + te.getMsg() + "
code: " + te.getErrorCode()
    );
  } catch (RemoteException re) {
        throw new RuntimeException("Unexpected RemoteException: " +
re);
      }
}
/**
 *    Perform a Texis related search query and package results (if
any)
 *  into an array of RelatedResults objects
 */
public RelatedResult[] findRelated(String rawQuery, String
canonQuery, int maxResults, int maxResultLength)
  throws Exception
{
  try {
    return
findRelated(rawQuery,canonQuery,maxResults,maxResultLength,2000);
    }catch (Exception e) {
    e.printStackTrace();
    throw new Exception("overloaded findRelated"+e.getMessage());
  }
}
public RelatedResult[] findRelated(String rawQuery, String
canonQuery, int maxResults, int maxResultLength, long timeOut)
    throws RelatedSearchException
{
  //local vars
  Vector resultVector = new Vector();
  Vector thisRow = new Vector();
  RelatedResult[] results = null;
  int resultCount = 0;
  int rank = 0;
  int canonCnt = 0;
  Integer advertiser_id = null;
  int advertiserCnt = 0;
  String advertiserIds = null;
  String rawSearchText = null;
  String canonSearchText = null;

  Runtime rt = Runtime.getRuntime();
  long mem = rt.totalMemory();
  long free = rt.freeMemory();
  //System.err.println("totalMemory(): " + mem);
  //System.err.println("freeMemory(): " + free);

  try {

    if (canonQuery == null) return(null);

    Vector queryArgs = new Vector();
```

```
String newQuery;

if (timeOut != 0){
  // get the loop out of wait() mode
  synchronized (watchDog){
    globalTimeOut = timeOut;
    //System.err.println(
    //  System.currentTimeMillis()
    //  + "-"
    //  + timeClock
    //  + " = "
    //  + (System.currentTimeMillis() - timeClock)
    //  + " Core: setting MID_QUERY");
    semaphore = MID_QUERY;
    timeClock = System.currentTimeMillis();
    // thread better be waiting on eternity
    watchDog.notify();
  }
}

// if no Raw Query, probably have canonicalized version only
if (rawQuery != null){
  // usual serving site, don't re-pluralize, just use
  // raw query;
  newQuery = stripNoiseChars(rawQuery);
}else{
  // only have a canonQuery to work with, so make
  // a rough approximation of a raw term to include in search
  // try and generate queries to cover up to MAX_PLURAL_QRY
possible
  // re-pluralized forms of the query
  newQuery = pluralize(stripNoiseChars(canonQuery));
} //if

if(newQuery == null) return null;
if(isAdult(newQuery)) return null;

// Set up the (stack allocated) query parameters
queryArgs.removeAllElements();
queryArgs.addElement(newQuery);

//perform JNI calls here
texisQuery.setParam(queryArgs);
texisQuery.execSQL();

//Iterate over the rows
String lastCanon = rawQuery;
Hashtable advertisers = new Hashtable(MAX_ROWS*200);
Hashtable used = new Hashtable(MAX_ROWS);
Vector resultSet = texisQuery.getRows();
//Vector resultSet = getRowsLocal();

// make 2 passes.
// first time de-dup on advertisers
// second time don't dedup
//System.out.println("got rows: " + resultSet.size());
for(int pass = 0; pass < 2; pass++){
    if(resultCount >= maxResults) break;
```

```
            for(int i = 0; i < resultSet.size(); i++){
          thisRow = (Vector)resultSet.elementAt(i);
          if(thisRow.size() == 6) {
            rank = ((Number)(thisRow.elementAt(0))).intValue();
//System.out.println(thisRow.elementAt(0).getClass().toString());

            canonCnt = ((Integer)(thisRow.elementAt(1))).intValue();


            rawSearchText = (String)thisRow.elementAt(2);
            canonSearchText = (String)thisRow.elementAt(3);
            advertiserIds = (String)thisRow.elementAt(4);
            advertiserCnt = ((Integer)thisRow.elementAt(5)).intValue();

            //Drop out early if we detect magic ADULT_FLAG
            if(advertiserCnt == ADULT_FLAG)  return null;
            if(canonCnt == ADULT_FLAG)  return null;

            if(false){
              System.out.println( "rank: "
              + rank
              + " cnt: "
              + canonCnt
              + " rst: "
              + rawSearchText
              + " cst: "
              + canonSearchText
              + " aids: "
              + advertiserIds
              + " adcnt: "
              + advertiserCnt
              );
            }
          }else{
            throw new RelatedSearchException("Texis query failed,
protocol violation");
          }
          // De-dup the results, and also don't return a related
          // search term which canonically matches the original query
          if ((!canonSearchText.equalsIgnoreCase(rawQuery)) &&
            (!canonSearchText.equals(canonQuery)) &&
            (!rawSearchText.equalsIgnoreCase(rawQuery)) &&
            (!rawSearchText.equalsIgnoreCase(canonQuery)) &&
            (canonSearchText.length() <= maxResultLength)){
            //System.out.println("got cst: " + canonSearchText);
            //look for this advertiser in the hash table
            //if there, increment occurrances count
            // and if above threshhold, we've seen enough
            // terms suggested by this advertiser, so go to
            // next term
            // if not seen this advertiser yet, put it in the
            // hashtable and process

            StringTokenizer st = new StringTokenizer(advertiserIds, " "
);
            //if(st.countTokens() != advertiserCnt){
            //  System.out.println("toks: "
```

```
            //    + st.countTokens()
            //    + "Cnt: "
            //    + advertiserCnt);
            //  throw new RelatedSearchException(
            //  "Texis query suspect, wrong advertiser count");
            //}
            if (pass == 0){

              int dupAdvCnt = 0;
              boolean Next = false;

              // Parse all the advertiser ID's out of the returned row
              while(st.hasMoreTokens()){
                Integer advertiserId = Integer.valueOf(st.nextToken());

                //if (!advertisers.containsKey(advertiserId))

                // if this advertiser is new to us (over whole query)
                // put in the hash
                Integer cnt = (Integer)advertisers.get(advertiserId);
                if (cnt == null)
                {
                  advertisers.put(advertiserId,new Integer(0));
                }else{
                  // Seen this advertiser before, so increment his
                  // tally
                  advertisers.put(advertiserId,new
Integer(cnt.intValue()+1));

                  //System.out.println(advertiserId + " dups: " +
(cnt.intValue()+1));
                  // If he's (now) past the threshhold, don't use
                  // bidded term (yet)
                  if (cnt.intValue() >= ADVERTISER_THRESHOLD){
                    Next = true;
                    break;
                  }
                  dupAdvCnt++;
                }
              }
              if (Next == true){
                continue;
              }else {
                if (!used.containsKey(canonSearchText)){
                  used.put(canonSearchText, new Boolean(true));
                }
              }
            }else {
              if (!used.containsKey(canonSearchText)){
                used.put(canonSearchText, new Boolean(true));
              }else{
                continue;
              }
            }

            //if (dupAdvCnt >= ADVERTISER_THRESHOLD){
              //continue;
            //  System.out.println("dups: " + dupAdvCnt);
```

```
        //}

    /**
            if (pass == 0){
              // first time thru see if we've used this advertiser
              Integer cnt = (Integer)advertisers.get(advertiser_id);
              if (cnt == null){
                advertisers.put(advertiser_id,new Integer(0));
              }else{
                advertisers.put(advertiser_id,new
    Integer(cnt.intValue()+1));
                if (cnt.intValue() >= ADVERTISER_THRESHOLD){
                  continue;
                }
              }
              if (!used.containsKey(canonSearchText)){
                used.put(canonSearchText, new Boolean(true));
              }
            }else{
              // this is a second (or more) time thru.
              // see if we've already used this term
              if (!used.containsKey(canonSearchText)){
                used.put(canonSearchText, new Boolean(true));
              }else{
                continue;
              }
            }
    **/
            if(resultCount < maxResults){
              resultVector.
                addElement(new RelatedResult(rawSearchText,
    RelatedResult.NON_CACHED));
              resultCount++;
            }else {
              break;
            }//if-else
          }//if
            }//for
        }//for
        }catch (TException te) {
        throw new RelatedSearchException("Texis interface failed with: "
    + te.getMsg(), te);
      }catch(Throwable t) {
        t.printStackTrace();
        throw new RelatedSearchException("Unexpected Texis failure with:
    " + t.getMessage(), t);
      }finally {
        if (timeOut != 0){
          synchronized (watchDog){
            //System.err.println(
            //  System.currentTimeMillis()
            //  + "-"
            //  + timeClock
            //  + " = "
            //  + (System.currentTimeMillis() - timeClock)
            //  + " Core: setting POST_QUERY");
            semaphore = POST_QUERY;
            timeClock = System.currentTimeMillis();
```

```
            // cause thread to wait on eternity
            globalTimeOut = 0;
            watchDog.notify();
            //System.err.println(
            //  System.currentTimeMillis()
            //  + "-"
            //  + timeClock
            //  + " = "
            //  + (System.currentTimeMillis() - timeClock)
            //  + " Core: done with calling notify");
          }
        }
      }

      //System.out.println("INFO: 115 err''s: " + cb.getErr115());
      if (cb.getErr115() > 100){
        System.out.println("FATAL: Too many Err115's");
        System.exit(3);
      }

      if(resultVector.size() == 0)
        return null;
      else {
        resultVector.copyInto(results = new
    RelatedResult[resultVector.size()]);
        return results;
      }
    }
    private String stripNoiseChars(String term){
    // Clean up the query a bit

        if(term.length() < 2) return(null);

        char[] buf = new char[term.length()];
        int firstChar = 0;

        term.getChars(0, buf.length, buf, 0);

        for (int i = 0; i < buf.length; i++){
          if (buf[i] < 0x20 || buf[i] > 0x7e) return(null);

          switch (buf[i]) {
          case ('~'):
          case ('`'):
          case ('!'):
          case ('@'):
          case ('#'):
          case ('$'):
          case ('%'):
          case ('^'):
          case ('&'):
          case ('*'):
          case ('('):
          case (')'):
          case ('-'):
          case ('_'):
          case ('+'):
          case ('='):
```

```
            case ('{'):
            case ('['):
            case ('}'):
            case (']'):
            case ('|'):
            case ('\''):
            case (':'):
            case (';'):
            case ('"'):
            case ('\\'):
            case ('>'):
            case (','):
            case ('<'):
            case ('.'):
            case ('/'):
            case ('?'):
            case (' '): {
              buf[i] = ' ';
              //System.out.println("i:"
              //+ i + "firstChar:" + firstChar
              //+ "setting buf[i]: "
              //+ (String.valueOf(buf[i])) + " setting to space");

              if(firstChar == i) firstChar = i+1;
            }
            }
          }
          // only spaces left
          if (firstChar == buf.length) return(null);

          term = term == null ? null :
String.valueOf(buf,firstChar,buf.length-firstChar).trim();

            switch(term.length()){
          case 0:
          case 1: return(null);
          default: {
            switch (buf[firstChar]){
            case ('h'):
            case ('H'):
            case ('w'):
            case ('W'):{
              String lowerTerm = term.toLowerCase();

              // Use the lcase vers of the string for testing
              // but make sure to SET the original string to return
              if(lowerTerm.startsWith("http   www")) term =
term.substring(10);
              else if(lowerTerm.startsWith("http  www")) term =
term.substring(9);
              else if(lowerTerm.startsWith("http www")) term =
term.substring(8);
              else if(lowerTerm.startsWith("hhttp   www")) term =
term.substring(11);
              else if(lowerTerm.startsWith("http ")) term =
term.substring(5);
              else if(lowerTerm.startsWith("http")) term =
term.substring(4);
```

```
        else if(lowerTerm.startsWith("www ")) term =
term.substring(4);
        else if(lowerTerm.startsWith("www")) term =
term.substring(3);
          }
        }
      }
      }
      switch (term.length()){
      case 0:
      case 1: return(null);
      default: {
        switch (term.charAt(term.length()-1)){
        case ('m'):
        case ('M'):
        case ('t'):
        case ('T'):
        case ('g'):
        case ('G'):
        case ('f'):
        case ('F'):{
          String lowerTerm = term.toLowerCase();

          if(lowerTerm.endsWith(" dot com")) term = term.length() > 8 ?
term.substring(0,term.length()-8) : null;
          else if(lowerTerm.endsWith(" dotcom")) term = term.length() >
7 ? term.substring(0,term.length()-7) : null;
          else if(lowerTerm.endsWith(" com")) term = term.length() > 4
? term.substring(0,term.length()-4) : null;
          else if(lowerTerm.endsWith(" net")) term = term.length() > 4
? term.substring(0,term.length()-4) : null;
          else if(lowerTerm.endsWith(" org")) term = term.length() > 4
? term.substring(0,term.length()-4) : null;
          else if(lowerTerm.endsWith(" gif")) term = term.length() > 4
? term.substring(0,term.length()-4) : null;
          else if(lowerTerm.endsWith(" jpg")) term = term.length() > 4
? term.substring(0,term.length()-4) : null;
          }
        }
      }
      }
      //Debug: System.out.println("term: [" + term.trim() + "]");
      return(term == null ? null : term.trim());
  }

  private boolean isAdult(String query)
      throws RelatedSearchException
  {
    if(query == null) return false;
    Vector queryArgs = new Vector();
    Vector thisRow = new Vector();
    queryArgs.addElement(query);

    try {
    //perform JNI calls
    texisAdultQuery.setParam(queryArgs);
    texisAdultQuery.execSQL();
    if((thisRow = texisAdultQuery.getRow()).size() != 0){
```

```
      return(true);
    }else{
      return(false);
    }
    }catch (TException te) {
    throw new RelatedSearchException("Texis interface failed with: " +
te.getMsg(), te);
    }catch (RemoteException re) {
          throw new RelatedSearchException("Got a RemoteException that
should never occur: " + re);
        }
}

private String pluralize(String token)
    throws RelatedSearchException
{
  if(token == null) return null;
  Vector queryArgs = new Vector();
  String pluralToken;
  Vector thisRow = new Vector();

  StringTokenizer st0 = new StringTokenizer(token, " " );
  String[] terms = new String[st0.countTokens()];
  String[] fullQuery = new String[MAX_PLURAL_QRY];
  int fullQueryCnt = 0;
  // Iterate over each token to see if there's a plural version
  for(int ele0 = 0; st0.hasMoreTokens(); ele0++){
    terms[ele0] = st0.nextToken();
  }

  for(int element = 0;  element < terms.length && fullQueryCnt <
MAX_PLURAL_QRY; element++){

    // Do plurals lookup on this term from texis db
    queryArgs.removeAllElements();
    queryArgs.addElement(terms[element]);

        try {
    //perform JNI calls
    texisPlurQuery.setParam(queryArgs);
    texisPlurQuery.execSQL();

    //retrieve the row
    if((thisRow = texisPlurQuery.getRow()).size() != 0){
      String term = null;
      // loop thru the terms
      for(int ele1 = 0; ele1 < terms.length; ele1++){
        if(ele1 == element){
            if(ele1 == 0){
          term = (String)(thisRow.elementAt(0));
            } else {
          term += " " + (String)(thisRow.elementAt(0));
            }
        }else{
            if(ele1 == 0)
          term = terms[ele1];
            else
          term += " " + terms[ele1];
```

```
            }
          }
          fullQuery[fullQueryCnt] = term;
          fullQueryCnt++;
        }
            }catch (TException te) {
        throw new RelatedSearchException("Texis interface failed with: "
+ te.getMsg(), te);
            }catch (RemoteException re) {
                    throw new RelatedSearchException("Got a
RemoteException that should never occur: " + re);}


      }

      // Build the new expanded query
      if(fullQueryCnt > 0){
        pluralToken = "(" + token;
        for(int i = 0; i < fullQueryCnt; i++){
          pluralToken = pluralToken + "," + fullQuery[i];
        }
        pluralToken = pluralToken + ")";
      }else{
        pluralToken = token;
      }
      return(pluralToken);
        }
        public Vector getRowsLocal() throws TException, RemoteException {
            Vector set = new Vector();
            int e;
            synchronized (APIToken.Lock) {
                while (true) {
                    Vector row = new Vector();
                    row = texisQuery.getRow();
                    if (row.size() == 0)
                        break;
                    set.addElement(row);
                }
            }
            return set;
        }

        public synchronized void run() {
        while (true){
      try{
      // start our timeout
      synchronized(watchDog){
        //System.err.println(
        //  System.currentTimeMillis()
        //  + "-"
        //  + timeClock
        //  + " = "
        //  + (System.currentTimeMillis() - timeClock)
        //  + " run: starting wait of "
        //  + globalTimeOut);
        watchDog.wait(globalTimeOut);
        // just got woke up,
        // see why
```

```
        if (semaphore.equals(PRE_QUERY)){
          //System.err.println(
          //  System.currentTimeMillis()
          //  + "-"
          //  + timeClock
          //  + " = "
          //  + (System.currentTimeMillis() - timeClock)
          //  + " run: got PRE_QUERY");
          continue;
        } else if (semaphore.equals(POST_QUERY)){
          //System.err.println(
          //  System.currentTimeMillis()
          //  + "-"
          //  + timeClock
          //  + " = "
          //  + (System.currentTimeMillis() - timeClock)
          //  + " run: got POST_QUERY");
          continue;
        } else if (semaphore.equals(MID_QUERY)){
          if (System.currentTimeMillis() - timeClock >= globalTimeOut){
            // we timed out, but semaphore wasn't
            // set, so hose ourselves
            System.err.println(
              System.currentTimeMillis()
              + "-"
              + timeClock
              + " = "
              + (System.currentTimeMillis() - timeClock)
              + " Fatal: timeout "
              + globalTimeOut
              + " usec exceeded");
            System.exit(1);
          }else{
            //System.err.println(
            //  System.currentTimeMillis()
            //  + "-"
            //  + timeClock
            //  + " = "
            //  + (System.currentTimeMillis() - timeClock)
            //  + " run: got MID_QUERY, but OK!");
          }
        } else {
          System.err.println(
            System.currentTimeMillis()
            + "-"
            + timeClock
            + " = "
            + (System.currentTimeMillis() - timeClock)
            + " run: ARGHH got no_QUERY, Hmmmmm !");
        }
      }
      }catch(Exception e){
        System.err.println("got wait() exception");
      }
        }
    }
    }
```

The following code is used to implement the cached results lookup, first to see if we've seen this related search before, to save time and not do the algorithmic lookup during the related search execution.

```
package com.go2.search.related;

//import atg.nucleus.GenericRMIService;
import atg.nucleus.GenericService;
import atg.nucleus.ServiceException;

import atg.service.resourcepool.JDBCConnectionPool;
import atg.service.resourcepool.ResourceObject;
import atg.service.resourcepool.ResourcePoolException;

import java.rmi.RemoteException;

import java.net.*;
import java.io.*;

import java.sql.*;

import java.util.Vector;

/**
*     This is the top level interface to the related search
*     system it is meant to be used as a dynamo service available
*     to other dynamo services
*
*/
//public class RelatedSearcherImpl extends GenericRMIService
public class RelatedSearcherImpl extends GenericService
                    implements RelatedSearcher
{
      //my pool of Texis/UDP
      private TexisUDPConnectionPool texisUDPConnectionPool;

      //my pool of connections to Oracle cache
      private JDBCConnectionPool relatedCacheConnectionPool;

      //Statistics properties
      private int             requestCount = 0;
      private int             oracleCacheHits = 0;
      private int             texisRequests = 0;
      private int             texisTimeoutMillis = 0;
      private int             slowTexisRequestCount = 0;

      //private constants
      private static String   CACHE_SQL = "SELECT * FROM REL_SEARCH
WHERE CANON_QUERY=?";
      private static int            BUFFER_SIZE = 512;

      //parameters
      private boolean   texisEnabled = false;
```

```
        private boolean   oracleEnabled = false;
        private boolean   systemEnabled = false;
        private long cummulativeOracleTime = 0;
        private long cummulativeTexisTime = 0;
/**
 * Create and export and instance of RelatedSearcher over RMI
 */
public RelatedSearcherImpl() throws RemoteException {
        super();

        //java.rmi.registry.LocateRegistry.createRegistry(1111).rebind(
"RelatedSearcher", this);
}
/**
 * This method was created in VisualAge.
 * @return RelatedResult[]
 * @param canonQuery java.lang.String
 * @param maxResults int
 * @param maxLength int
 */
private RelatedResult[] findFromCache(String canonQuery, int
maxResults, int maxLength)
        throws RelatedSearchException
{
        Vector resultVector = new Vector();
        RelatedResult[] results = null;
        PreparedStatement ps = null;
        ResultSet rs = null;
        try {
                //Get a Connection
                ResourceObject resource = null;
                try {
                        resource = getRelatedCacheConnectionPool().checkOut
(getAbsoluteName ());
                        Connection conn = (Connection) resource.getResource
();
                        boolean success = false;
                        try {
                                //Here's where we get the goods from Oracle
                                ps = conn.prepareStatement(CACHE_SQL);
                                ps.setString(1, canonQuery);
                                rs = ps.executeQuery();
                                //prime the cursor to point to the one and
only row we
                                //expect from Oracle if now matching rows
were found
                                //then we'll simply drop thru to the end
                                if(rs.next()){
                                        //Extract the data we need if there was
something
                                        int numTerms = rs.getInt(2);
                                        if(numTerms == 0)
                                                //The cache tells us that there
won't be
                                                //any results so we'll bail early
                                                throw new
RelatedSearchException("No related Results");
                                        int cacheFlag = rs.getInt(3);
```

```
                                    //iterate over results retrieving upto
maxResults
                                    //of those of them that are maxLength
or smaller
                                    int resultCount = 0, rowCount = 0;
                                    while(resultCount < maxResults &&
rowCount < numTerms) {
                                          String term = rs.getString(4 +
rowCount);
                                          //push this term into the result
vector if its good
                                          if(term.length() <= maxLength) {
                                                resultVector.addElement(new
RelatedResult(term, cacheFlag));
                                                resultCount++;
                                          }
                                          rowCount++;
                                    }
                              }
                        conn.commit();
                              success = true;
                        }finally{
                              //Cleanup result set
                              if(rs != null)
                                    rs.close();
                              //Cleanup prepared statement
                              if(ps != null)
                                    ps.close();
                              //Cleanup connection
                              if (!success && conn != null) conn.rollback
();
                        }//try-finally
                  }finally {
                        // Check the Connection back in
                        if (resource != null)
                              getRelatedCacheConnectionPool().checkIn
(resource);
                  }//try-finally
            }catch (ResourcePoolException exc) {
                  if (isLoggingError ()) {
                        logError ("Unable to get Oracle cache connection",
exc);
                  }
                  throw new RelatedSearchException("Unable to get Oracle
cache connection", exc);
            }catch (SQLException se) {
                  if (isLoggingError ()) {
                        logError ("Interface with Oracle cache failed",
se);
                  }
                  throw new RelatedSearchException("Interface with Oracle
cache failed", se);
            }//try-catch
            if(resultVector.size() == 0)
                  return null;
            else {
                  resultVector.copyInto(results = new
RelatedResult[resultVector.size()]);
```

```
                return results;
        }
}
/**
*       Communicate to Texis thru TexisConnectionPool
*       @return RelatedResult[]
*       @param canonQuery java.lang.String
*       @param maxResults int
*       @param maxLength int
*/
private RelatedResult[] findFromUDPTexis(String rawQuery, String
canonQuery, int maxResults, int maxLength)
        throws RelatedSearchException
{

        RelatedResult[] results = null;
        //Get a UDPTexisConnection
        ResourceObject resource = null;
        TexisUDPConnection tc  = null;

        try {
                resource = getTexisUDPConnectionPool().checkOut
(getAbsoluteName ());
                tc = (TexisUDPConnection) resource.getResource ();
                DatagramSocket socket = tc.getSocket();

                //do this at run time to be able to switch Dynamo at run
time
                socket.setSoTimeout(getTexisTimeoutMillis());

                //package data to send
                TexisRequest request = new TexisRequest();
                request.setRawQuery(rawQuery);
                request.setCanonQuery(canonQuery);
                request.setMaxResults(maxResults);
                request.setMaxChars(maxLength);
                request.setSequenceNumber(++tc.sequenceNumber);
                request.setTimeout(getTexisTimeoutMillis());

                ByteArrayOutputStream baos = new ByteArrayOutputStream();
                ObjectOutputStream ous = new ObjectOutputStream(baos);
                ous.writeObject(request);
                ous.flush();
                baos.close();
                byte[] sendData = baos.toByteArray();
                //send it off to the server
                if(isLoggingDebug()) {
                        logDebug("About to send to Texis at endpoint: " +
tc.getHost() + ":" + tc.getPort());
                }
                //send it
                DatagramPacket sendPacket = new DatagramPacket(sendData,
sendData.length, tc.getHost(), tc.getPort());
                socket.send(sendPacket);

                //wait for a reply upto timeOut milliseconds
                long startWait = System.currentTimeMillis();
```

```
            while(true) {  //pull off inboud packets and check them
the the right sequenceNumber
                        //throws a java.io.InterruptedIOExeption on timeout
                        DatagramPacket receivePacket = new
DatagramPacket(new byte[BUFFER_SIZE], BUFFER_SIZE);
                        socket.receive(receivePacket);
                        ObjectInputStream ois = new ObjectInputStream(new
ByteArrayInputStream(receivePacket.getData()));
                        TexisResponse response =
(TexisResponse)ois.readObject();
                        ois.close();
                        if(response.getSequenceNumber() !=
tc.sequenceNumber) {
                              //we got a stale response
                              long midPoint = System.currentTimeMillis();
                              int remainder = (int)(getTexisTimeoutMillis()
- (midPoint - startWait));
                              if(remainder > 0) { //if we can still wait
some more before a timeout
                                    //reset socket timeOut to the remaining
time
                                    socket.setSoTimeout(remainder);
                              }else {
                                    //give up at this point
                                    break;
                              }
                        }//if-wrong-sequence-number
                        else {
                              results = response.getResults();
                              break;
                        }
                  }//while

            }catch (ResourcePoolException rpe) {
                  if (isLoggingError ()) {
                        logError ("Unable to get or checkin a Texis
connection", rpe);
                  }
            }catch(ClassNotFoundException cnfe) {
                  if (isLoggingError ()) {
                        logError ("Class not found Exception", cnfe);
                  }
            }catch(SocketException se) {
                  if (isLoggingError ()) {
                        logError ("Socket Exception talking to Texis", se);
                  }
            }catch(StreamCorruptedException sce) {
                  if (isLoggingError ()) {
                        logError ("Corrupted return from Texis", sce);
                  }
            }catch(InterruptedIOException ioie) {
                  if (isLoggingDebug()) {
                        logDebug("Timed out talking to Texis", ioie);
                  }
            }catch(IOException ioe) {
                  if (isLoggingDebug()) {
                        logDebug("Timed out talking to Texis", ioe);
                  }
```

```
        }
        finally {
              // Check the Connection back in if we got it in the first
place
              try {
                    if (resource != null)
                          getTexisUDPConnectionPool().checkIn
(resource);
              }catch(ResourcePoolException rpe){  /* ignore this one */
}
        }

        return results;
}
/**
 * @return RelatedResult[] - an  array of RelatedResult objects
 * which is ordered by relebance from high to low or null if the
system is disable or no
 * related results were found
 * @param rawQuery java.lang.String - raw query for which related
searches are needed
 * @param canonQuery java.lang.String - canonocalized for of the raw
query
 * @param maxResults int - maximum number of results requested
 * @param maxResultLenght int - maximum lenght of a result in
characters
 */
public RelatedResult[] findRelated(String rawQuery, String
canonQuery, int maxResults, int maxResultLength)
                                                //throws
RelatedSearchException
                                                throws
RelatedSearchException, RemoteException
{

        requestCount++;
        //Return fast if system is disabled
        if(!getSystemEnabled())
              return null;

        RelatedResult[] results = null;

        //first try getting data from the Oracle pool (if enabled)
        if(getOracleEnabled()) {
              try {
                    long startOracle = System.currentTimeMillis();
//keep timing stats
                    results = findFromCache(canonQuery, maxResults,
maxResultLength);
                    oracleCacheHits++; //fixed statistics bug
                    cummulativeOracleTime +=
(System.currentTimeMillis() - startOracle);
              }catch(RelatedSearchException rse) {
                    //If Oracle told us that this search has no related
                    //i.e. editorially-excluded porn, then drop out
early
                    if(rse.getRootCause() == null) {
                          return null;
```

```
                    }else { //log it otherwise for post mortem
                        if(isLoggingError())
                            logError("Failed to interface to Oracle
cache, will try Texis", rse);
                    }
                }catch(Exception e) {
                    if(isLoggingError())
                        logError("Failed to interface to Oracle", e);
                }

        }//if Oracle enabled

        //if unsuccessfull then try Texis pool if enabled
        if(getTexisEnabled() && results == null) {
            try {
                long startTexisQuery = System.currentTimeMillis();
//keep texis timing stats
                texisRequests++;
                results = findFromUDPTexis(rawQuery, canonQuery,
maxResults, maxResultLength);
                long texisQueryMillis = System.currentTimeMillis()
- startTexisQuery;
                //log abnormally long request time
                if(texisQueryMillis > getTexisTimeoutMillis())
                    slowTexisRequestCount++;
                cummulativeTexisTime += texisQueryMillis;
            }catch(Exception e){
                if(isLoggingError())
                    logError("Texis interface failed with: " +
e.getMessage() , e);
            }
        }//if texisEnabled


        return results;
    }
    /**
     * Stats accessor
     * @return String
     */
    public String getCummulativeOracleTime() {
            return (cummulativeOracleTime / 1000.0) + " seconds";
    }
    /**
     * Stats accessor
     * @return long
     */
    public String getCummulativeTexisTime() {
        return (cummulativeTexisTime / 1000.0) + " seconds";
    }
    /**
     * Stats accessor
     * @return int
     */
    public int getOracleCacheHits() {
        return oracleCacheHits;
    }
    /**
```

```
     * Stats accessor
     * @return boolean
     */
    public boolean getOracleEnabled() {
          return oracleEnabled;
    }
    /**
     * Accessor for relatedCacheConnectionPool
     * @return atg.service.resourcepool.JDBCConnectionPool
     */
    public JDBCConnectionPool getRelatedCacheConnectionPool() {
          return relatedCacheConnectionPool;
    }
    /**
     * Stats accessor
     * @return int
     */
    public int getRequestCount() {
          return requestCount;
    }
    /**
     * Stats accessor
     * @return int
     */
    public int getSlowTexisRequestCount() {
          return slowTexisRequestCount;
    }
    /**
     * Stats accessor
     * @return boolean
     */
    public boolean getSystemEnabled() {
          return systemEnabled;
    }
    /**
     * Stats accessor
     * @return boolean
     */
    public boolean getTexisEnabled() {
          return texisEnabled;
    }
    /**
     * stats accessor
     * @return int
     */
    public int getTexisRequests() {
          return texisRequests;
    }
    /**
     * configu param accessor
     * @return int
     */
    public int getTexisTimeoutMillis() {
          return texisTimeoutMillis;
    }
    /**
     * This method was created in VisualAge.
     * @return com.go2.search.related.TexisUDPConnectionPool
```

```
 */
public TexisUDPConnectionPool getTexisUDPConnectionPool() {
      return texisUDPConnectionPool;
}
/**
 * mutator
 * @param newValue boolean
 */
public void setOracleEnabled(boolean newValue) {
      this.oracleEnabled = newValue;
}
/**
 * Mutator for relatedCacheConnectionPool
 * @param newValue atg.service.resourcepool.JDBCConnectionPool
 */
public void setRelatedCacheConnectionPool(JDBCConnectionPool
newValue) {
      this.relatedCacheConnectionPool = newValue;
}
/**
 * mutator
 * @param newValue boolean
 */
public void setSystemEnabled(boolean newValue) {
      this.systemEnabled = newValue;
}
/**
 * mutator
 * @param newValue boolean
 */
public void setTexisEnabled(boolean newValue) {
      this.texisEnabled = newValue;
}
/**
 * parameter mutator
 * @param newValue int
 */
public void setTexisTimeoutMillis(int newValue) {
      this.texisTimeoutMillis = newValue;
}
/**
 * This method was created in VisualAge.
 * @param newValue com.go2.search.related.TexisUDPConnectionPool
 */
public void setTexisUDPConnectionPool(TexisUDPConnectionPool
newValue) {
      this.texisUDPConnectionPool = newValue;
}
}
```

The following code is control code that controls the dumping of search
listing database, loading the crawled text, and inverted-indexing
all the related search indexing, including building the 'derived-data' elements:

```
#!/bin/ksh -x
export PATH=/usr/local/morph3/bin:.:$PATH
#. ./.zshrc

export TMP=/export/home/goto/tmp
export TEMP=$TMP
export TEMPDIR=$TMP
export TMPDIR=$TMP

TMPTABLE=line_ad0
TMPTABLE2=line_ad
TERMSTABLE=terms

INC=02
NEWTABLE=line_ad${INC}

CRAWLDATA=/home/goto/rs/DONE/ALL.UNIQ
CRAWLTABLE=line_ad4
SPOOL=/home/goto/list

DB=/home/goto/crawldb

################
Log(){
      echo '\n####' $(date "+%m/%d %H:%M:%S"): "${*}" '####'
}

log 0. timport crawled data
Log 0.1 Create line_ad4 and unique index in preparation for 'crawl'
import

tsql -d $DB <<!
drop table line_ad4;
create table line_ad4(
 id counter,
 ad_url varchar(300),
 crawltitle varchar(750),
 crawlmeta varchar(500),
 crawlbody varchar(8000)
);
drop index idx4ad_url;
create unique index idx4ad_url on line_ad4(ad_url);
!

timport -database $DB -table $CRAWLTABLE -s
/home/goto/rs/DONE/crawl.sch -file $CRAWLDATA
```

```
Log 1. extract line_ads from live_ADMN into column delimited spool
file
umpadm $SPOOL

Log 2. timport  -database $DB -table $TMPTABLE -s newrs.sch -file
${SPOOL}
timport  -database $DB -table $TMPTABLE -s newrs.sch -file ${SPOOL}

Log 3. build canon index on $TMPTABLE
##
tsql -d ${DB} <<!
drop index idx0cst;
create index idx0cst on ${TMPTABLE}(cannon_search_text);
!

##
Log 4. add counts of canons to $TMPTABLE
texis DB=$DB TMPTABLE=$TMPTABLE updatecnt

Log 5. build url index on $TMPTABLE
tsql -d ${DB} <<!
drop index idx0url;
create index idx0url on ${TMPTABLE}(ad_url);
!

Log 6. merge crawled text w/ original
tsql -d ${DB} <<!
drop index idx4url;
create index idx4url on ${CRAWLTABLE}(ad_url);
drop table $TMPTABLE2;
CREATE TABLE $TMPTABLE2
AS
SELECT
a.price price,
a.rating rating,
a.ad_id ad_id,
a.bid_date bid_date,
a.raw_search_text raw_search_text,
a.cannon_search_text cannon_search_text,
a.ad_spec_title ad_spec_title,
a.ad_spec_desc ad_spec_desc,
a.ad_url ad_url,
a. resource_id resource_id,
b.crawltitle crawltitle,
b.crawlmeta crawlmeta,
b.crawlbody crawlbody,
a.canon_cnt
from $TMPTABLE a, $CRAWLTABLE b
where a.ad_url = b.ad_url
order by price desc;

!

#texis DB=$DB CRAWLTABLE=$CRAWLTABLE TMPTABLE=$TMPTABLE updateit

Log 7. collapse 0 onto 01
Log 7.1 first make the table
```

```
tsql -d ${DB} <<!
drop table ${NEWTABLE};
!

tsql -d ${DB} <<!
create table ${NEWTABLE}(
canon_cnt               integer,
cannon_search_text       varchar(50),
raw_search_text         varchar(50),
advertiser_ids          varchar(4096),
advertiser_cnt          integer,
words                   varchar(65536)
);
!

Log 7.2 second, build the uniq, sorted list of search terms
Log 7.2 select cannon_search_text from ${TMPTABLE};
tsql -d ${DB} <<! | sort -u | timport -s termstable.sch -file -
select cannon_search_text from ${TMPTABLE};
!

Log 7.3 third, build uniq index on terms table
Log 7.3 create unique index idxterm on terms term
tsql -d ${DB} <<!
create unique index idxterm on terms(term);
!

Log 7.3.9 prepare for collapse
Log 7.3.9 create index idxcstcol on $TMPTABLE2 cannon_search_text
Log 7.3.9 create index idxadidcol on $TMPTABLE2 ad_id
tsql -d ${DB} <<!
create index idxcstcol on $TMPTABLE2 (cannon_search_text);
create index idxadidcol on $TMPTABLE2 (ad_id);
!
Log 7.4 fourth collapse around csts
Log 7.4 texis SRCTABLE=$TMPTABLE2 TGTTABLE=$NEWTABLE
TERMSTABLE=$TERMSTABLE collapse
texis db=$DB SRCTABLE=$TMPTABLE2 TGTTABLE=$NEWTABLE
TERMSTABLE=$TERMSTABLE collapse

Log 8 do the porn line
buildporn | timport -database $DB -table $NEWTABLE -s rsporn.sch -
file -

Log 9 do the porn table
newporn | timport -database $DB -s rsnewporn.sch -file -

Log 10 metamorph index words column
tsql -d $DB <<!
create metamorph inverted index mmx${INC}w on ${NEWTABLE}(words);
!
Log 10 all done
```

```
Dumps bid-for-placement search listings data

#!/bin/ksh

TXSORAUSER=XXXXXXX
TXSORAPWD=XXXXXXX
LVSRVPWD=XXXXXX
#SPOOL=pipe
SPOOL=${1}
SERVER=XXXXXX


Log(){
      echo '\n####' $(date "+%m/%d %H:%M:%S"): "${1}" '####'
}

Log "start dump"

sqlplus -S ${TXSORAUSER}/${TXSORAPWD}@l${SERVER} >/dev/null <<!
      //set heading off
      set linesize 750
      set pagesize 0
      set arraysize 1
      set maxdata 50000
      set buffer 50000
      set crt off;
      set termout off
      spool ${SPOOL}
      select
            rpad(to_char(advertiser_id),8)||
            rpad(raw_search,30)||
            rpad(canon_search,30)||
            rpad(title,100)||
            rpad(description,280)||
            rpad(url,200)||
            rpad(resource_id,20)||
            rpad(to_char(price*100),5)||
            rpad(rating,2)||
            rpad(to_char(search_id),8)||
            rpad(resource_id,18)||
            rpad(to_char(line_ad_id),8)||
            to_char(bid_date, 'YYYYMMDD HHMMSS')
      from ads
      where status = 5 and rating = 'G' and canon_search <> 'grab
bag'
      and rownum < 10000;
      spool off;
      quit;
!
Log "end dump"
exit 0
```

Counts # of occurances of pagebids for each particular potential related search result

```
<script language=vortex>
<timeout = -1></timeout>
<a name=main>
<DB = "/home/goto/crawldb">
<SQL ROW "select distinct cannon_search_text cst from " $TMPTABLE>
  <SQL ROW "select count(*) cnt from " $TMPTABLE " where
cannon_search_text = $cst">
    <SQL NOVARS "update " $TMPTABLE " set canon_cnt = $cnt where
cannon_search_text = $cst" >
    </SQL>
  </SQL>
</SQL>
</a>
</script>
```

Aggregates web page body-text and listings based on the related-search result, while collecting and creating derived-data of 1, how many different advertisers have web-pages associated with the related-search result.

```
<script language=vortex>
<timeout = -1></timeout>
<DB = /home/goto/crawldb>
<a name=main>
<!-- get all canon-terms from tmp table -->

<SQL ROW "select term cst from " $TERMSTABLE >

  <$words =>
  <$rsts =>
  <$csts =>
  <$asts =>
  <$asds =>
  <$cts =>
  <$cms =>
  <$cbs =>
  <$advs =>
  <$last_adv =>
  <$adv_cnt = 0>
  <!-- get all rows w/ this canon term from tmp table -->
  <SQL ROW
    "select canon_cnt cc, ad_id aid, raw_search_text rst,
cannon_search_text cts, ad_spec_title ast,
     ad_spec_desc asd, crawltitle ct, crawlbody cb, crawlmeta cm
     from " $SRCTABLE " where cannon_search_text = $cst order by
ad_id">
    <!-- aggregate the text to prepare for collapsed insert -->
    <$rsts = ( $rsts + ' ' + $rst ) >
    <$rsts = ( $csts + ' ' + $cst ) >
    <$asts = ( $asts + ' ' + $ast ) >
    <$asds = ( $asds + ' ' + $asd ) >
    <$cts = ( $cts + ' ' + $ct ) >
    <$cms = ( $cms + ' ' + $cm ) >
    <$cbs = ( $cbs + ' ' + $cb ) >

    <if $aid != $last_adv>
      <!-- add advertiser to list if not seen him before -->
      <$advs = ( $advs +  ' ' + $aid ) >
      <$adv_cnt = ( $adv_cnt + 1 ) >
      <$last_adv = $aid>
    </if>
  </SQL>
  <$canon_cnt = $loop>
  <$words = ( $rsts + ' ' + $csts + ' ' + $asts + ' ' + $asds + ' ' +
$cms + ' ' + $cbs + ' ' + $cts ) >
  <!-- pick off zeroeth element only from $rst array -->
```

```
  <loop $rst>
    <$Rst = $rst>
    <break>
  </loop>
  <strlen $words>
  <$wlen = $ret>
  <strlen $advs>
  <!-- display which row we're working on -->
  $wlen  $ret  $cst
  <!-- insert to collapsed row -->
  <SQL NOVARS "insert into " $TGTTABLE " (canon_cnt,
cannon_search_text, raw_search_text, advertiser_ids, advertiser_cnt,
words) VALUES ($cc, $cst, $Rst, $advs, $adv_cnt, $words)" ></SQL>
  <!--$words ***<$ret = (text2mm ($words, 50))> $ret -->
</SQL>
</a>
</script>
```

Database schema layout used to upload bidded search listings

```
database /home/goto/crawldb
#droptable line_ad1
droptable line_ad0
table line_ad0
createtable
col
#keepfirst
trimspace
#multiple
datefmt  yyyymmdd HHMMSS

#       Name                    Type            Tag
default_val
field   advertiser_id           varchar(8)      1-8
""
field   raw_search_text         varchar(40)     9-48
""
field   cannon_search_text      varchar(40)     49-88
""
field   ad_spec_title           varchar(100)     89-188
""
field   ad_spec_desc            varchar(2000)   189-2188
""
field   ad_url                  varchar(200)    2189-2388
""
field   resource_id             varchar(20)     2389-2408
""
field   price                   integer         2409-2413
0
field   rating                  char(2)         2414-2415
""
field   ad_id                   integer         2416-2423
0
field   bid_date                date            2424-2438
0
field   canon_cnt       integer           -                 0
field   crawlwords              varchar(40)     -
""
```

Manual join of search listing data with crawled web page data into a single merged table

```
<script language=vortex>
<timeout = -1></timeout>
<a name=main>
<DB = "/home/goto/crawldb">
<SQL ROW "select ad_url myurl, crawltitle ct, crawlmeta cm, crawlbody
cb from " $CRAWLTABLE>
 <SQL NOVARS "update " $TMPTABLE " set crawltitle = $ct, crawlmeta =
$cm, crawlbody = $cbwhere ad_url = $myurl" >
 </SQL>
</SQL>
</a>
</script>
```

Code to duplicate URL Crawl elimination

```
/**
 * Insert the type's description here.
 * Creation date: (02/18/2000 11:12:12 AM)
 * @author:
 */
import java.io.*; //import of java classes needed for input/output
import java.util.*;
// import corejava.*;
import java.lang.String;
public class Url {
/*********************************************************************
 * Compare URLs Address                                              *
 *********************************************************************
 */
 public static void main(String args[]) throws Exception
 {

// Decalarations of the input and output File
      BufferedReader inputFile;
      PrintWriter nonDupFile;
      PrintWriter dupFile;

// Initialization
      String firstUrl="";
      String secondUrl="";
      String urlBufferA, urlBufferB="", urlBufferC="";

      String compareDomainA ="";
      String compareDomainB="";
      String compareDomainC="";
      String newFlag="false";
      inputFile = new BufferedReader(new
FileReader("/home/lauw/urls.lau"));


      nonDupFile = new PrintWriter(new
FileWriter("/home/lauw/nonDupFile.real"));
        dupFile = new PrintWriter(new
FileWriter("/home/lauw/dupFile.real"));

        nonDupFile.close();
        dupFile.close();

      firstUrl=inputFile.readLine();
      secondUrl=inputFile.readLine();
      urlBufferC=inputFile.readLine();

      urlBufferA= firstUrl;
      urlBufferB= secondUrl;

do

{

        Slash ccompareDomainA = new Slash();
```

```
        Slash ccompareDomainB = new Slash();
        Slash ccompareDomainC = new Slash();

        compareDomainA = ccompareDomainA.Slash(urlBufferA);
        compareDomainB = ccompareDomainB.Slash(urlBufferB);
        compareDomainC = ccompareDomainC.Slash(urlBufferC);

        Compare compareSub = new Compare();

          newFlag=compareSub.Compare(compareDomainA, compareDomainB,
compareDomainC, urlBufferB, newFlag);
          urlBufferA=urlBufferB;
        urlBufferB=urlBufferC;
        urlBufferC=inputFile.readLine();

        }while(urlBufferC!=null);
//////////////////////////Loop for first Null value

        urlBufferC=firstUrl;

        Slash ccompareFirstNullDomainA = new Slash();
          Slash ccompareFirstNullDomainB = new Slash();
          Slash ccompareFirstNullDomainC = new Slash();

          compareDomainA = ccompareFirstNullDomainA.Slash(urlBufferA);
          compareDomainB = ccompareFirstNullDomainB.Slash(urlBufferB);
          compareDomainC = ccompareFirstNullDomainC.Slash(urlBufferC);

          Compare compareFirstNullSub = new Compare();

          newFlag=compareFirstNullSub.Compare(compareDomainA,
compareDomainB, compareDomainC, urlBufferB, newFlag);
//////////////////////////Loop for last Null value
urlBufferA=urlBufferB;
urlBufferB=firstUrl;
urlBufferC=secondUrl;

        Slash ccompareLastNullDomainA = new Slash();
          Slash ccompareLastNullDomainB = new Slash();
          Slash ccompareLastNullDomainC = new Slash();

          compareDomainA = ccompareLastNullDomainA.Slash(urlBufferA);
          compareDomainB = ccompareLastNullDomainB.Slash(urlBufferB);
          compareDomainC = ccompareLastNullDomainC.Slash(urlBufferC);


          Compare compareLastNullSub = new Compare();

          newFlag=compareLastNullSub.Compare(compareDomainA,
compareDomainB, compareDomainC, urlBufferB, newFlag);
        inputFile.close();
        }
}
```

```
class Slash{

String Slash(String buffer)
{
              int  domainSlashEnd =0;
      int domainSlashStart = 0;
         boolean domainIndex = false;
         boolean startFound = false;
           boolean newFlag = false;
         String comparedomain;
         comparedomain="";

for (int domainSlashLoop=8; domainSlashLoop <= (buffer.length()-1);
domainSlashLoop++)
{if
((buffer.substring(domainSlashLoop,(domainSlashLoop+1)).equals("/"))
||
(buffer.substring(domainSlashLoop,(domainSlashLoop+1)).equals("?")))
 {
            if (startFound==false)
            {
            /// Check the Urls with Domain Name only
            if ((domainSlashLoop + 1)==buffer.length())
                  {

      comparedomain=buffer.substring(0,(buffer.length()));
                  domainIndex=true;
                  domainSlashLoop=buffer.length() + 500;
                  }//end for domain name only
            domainSlashStart = domainSlashLoop + 1;
            startFound = true;
            }
            else
            {
                  domainSlashEnd=domainSlashLoop;
                   domainSlashLoop = buffer.length() + 500; ///add 5
to get out of the loop
            }

 }// end for Loop
if (domainSlashEnd==0)
 {
       domainSlashEnd = buffer.length();
 }
if (domainIndex==false)
 {
      comparedomain=buffer.substring(domainSlashStart,
domainSlashEnd);
 }
}
return comparedomain;
}
}
```

```
import java.io.*; //import of java classes needed for input/output
class  Compare
{

      String Compare (String aCompareDomainA, String aCompareDomainB,
String aCompareDomainC, String aUrlBufferB, String newFlag) throws
Exception
{
      PrintWriter nonDupFile;
        PrintWriter dupFile;


      nonDupFile = new PrintWriter(new
FileWriter("/home/lauw/nonDupFile.real", true),true);
        dupFile = new PrintWriter(new
FileWriter("/home/lauw/dupFile.real",true),true);
      if ( aCompareDomainC.equals(aCompareDomainB))

      {
            if (newFlag.equals("true"))
                  {
                        dupFile.println("New");
                        newFlag = "false";
                  }
            System.out.println("Duplicate");
            dupFile.println(aUrlBufferB);
      }
      else
      {
      if (aCompareDomainB.equals(aCompareDomainA))

      {
            if (newFlag.equals("true"))
                  {
                        dupFile.println("New");
                        newFlag = "false";
                  }
            System.out.println ("print a Duplicat in second time");
            System.out.println("Sec Duplicate");
            dupFile.println(aUrlBufferB);
      }
      else
      {
            System.out.println( "non Dup");
            nonDupFile.println(aUrlBufferB);
            newFlag="true";
      }
      System.out.println("**************************************");

}

return newFlag;

}
}
```